MASTER IN COMPUTER SCIENCE

Performance of different TCP versions over UMTS common/dedicated channels

Dubois, Xavier

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix - University of Namur
Institut d'Informatique - Computer Science Institute

Performance of Different TCP Versions over UMTS Common/Dedicated Channels

Xavier Dubois

Supervised by

Laurent Schumacher - Troels B. Sorensen - Oumer M. Teyeb

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Année académique 2004-2005

Abstract

This report deals with the analysis of the performance of different versions of the wide spread Transport Control Protocol (TCP) over 3G UMTS networks. This analysis is done through simulations executed on the Network Simulator version 2 (NS2) and its Enhanced UMTS Radio Access Network Extension (EURANE). Different tool enhancements were developed in order to make this project possible.

This paper starts with explaining the main UMTS networks and TCP functionalities. Then, the performance of the most common TCP versions is analyzed with data transfer on the common channels only. Afterwards, the case of WEB traffic with a possible dedicated channel allocation is described. The impact of several parameters is analyzed. This last set of simulations is first done using a single non-preemptive inactivity timer for the DCH de-allocation, and then with hybrids preemptive/non-preemptive inactivity timers.

Résumé

Ce rapport tente d'analyser les performances de différentes versions du protocole de transport TCP (Transport Control Protocol) sur les réseaux 3G UMTS. Cette analyse est menée via des simulations utilisant le Network Simulator version 2 (NS2) et son extension UMTS (Enhanced UMTS Radio Access Network Extension, EURANE). Différents utilitaires ont dû être développés pour rendre ce projet possible.

Cette thèse commence par expliquer les principales fonctionnalités de l'UMTS et de TCP. La performance des versions les plus communes de TCP est ensuite analysée via un transfert de données sur les canaux communs. Ensuite, Le cas d'un trafic WEB avec l'allocation possible de canaux dédicacés est décrite et l'impact de différents paramètres est analysé. Ce dernier jeu de simulations utilise d'abord un *inactivity timer* non préemptif pour la désallocation d'un canal dédicacé et ensuite un système mixte d'*inactivity timer* non préemptif et préemptif.

Preface

This report is mainly the result of a 9th Semester project in Mobile Communication, carried out in Fall 2004 in the Aalborg University (Denmark) and continued in the University of Namur (Belgium) in spring 2005.

The structure of the report does not exactly reflect the way this project was conducted. The analysis and documentation about UMTS and TCP came first. Afterwards, most of the simulations about data transfer over the common channels were done. Afterwards, it seemed important to implement some new features to the used tools to make this project possible. Then the simulations concerning the web traffic and a single preemptive inactivity timer for the dedicated channels allocation could run. Finally, a new hybrid inactivity timer mechanism was implemented for the last web traffic simulations.

I would like to thank my Belgian and Danish supervisors, Laurent Schumacher, Troels B. Sorensen and Oumer Teyeb for their guidance during the project and for their help in writing this report.

Xavier Dubois
University of Namur
June 2005

Abbreviations

3GPP	3rd Generation Partnership Project
ACK	Acknowledgement
AM	Acknowledged Data Transfer Service at the RLC
ARQ	Automatic Repeat Request
ARQ	Automatic Repeat Request
AS	Access Stratum
AWND	Advertised window
BCCH	Broadcast Control Channel
BCH	Broadcast Channel
BER	Bit Error Ratio
BLER	Block Error Rate
BMC	Broadcast/Multicast Control
BSS	Base Station Subsystem
CBR	Constant Bit Rate
CBS	Cell Broadcast Service
CCCH	Common Control Channel
CDMA	Code Division Multiple Access
CN	Core Network
CPCH	Common Packet Channel
CS	Circuit Switched
CTCH	Common Traffic Channel
CWND	Congestion window
DCCH	Dedicated Control Channel
DCH	Dedicated Channel
DPCCH	Dedicated Physical Control Channel
DPDCH	Dedicated Physical Data Channel
DRNC	Drift RNC
DSCH	Downlink Shared Channel
DTCH	Dedicated Traffic Channel
EURANE	Enhanced UMTS Radio Access Network Extensions for NS2
FACH	Forward Access Channel
FEC	Forward Error Correction
GGSN	Gateway GPRS Support Node

GMSC	Gateway MSC
GPRS	General Packet Radio System
GSM	Global System for Mobile communication
HHO	Hard Handovers
HLR	Home Location Register
HO	Handovers
HS-DSCH	High Speed Downlink Shared Channel
IP	Internet Protocol
ISN	Initial Sequence Number
MAC	Medium Access Control
ME	Mobile Equipment
MRW	Move Receiving Window
MSC	Mobile Switching Centre
MSS	Maximum Segment Size
NAS	Non Access Stratum
OVSF	Orthogonal Variable Spreading Factor
PC	Power Control
PCCH	Paging Control Channel
P-CCPCH	Primary Common Control Physical Channel
PCH	Paging Channel
PCPCH	Physical Common Packet Channel
PDCP	Packet Data Converge Protocol
PDU	Protocol Data Unit
PDSCH	Physical downlink Shared Channel
PICH	Paging Indicator Channel
PRACH	Physical Random Access Channel
PS	Packet Switched
PSTN	Public Switched Telephone Network
RACH	Random Access Channel
RAT	Radio Access Technologies
RLC	Radio Link Control
RNC	Radio Network Controller
RNS	Radio Network Subsystem
RRC	Radio Resource Control
RRM	Radio Resource Management
RTO	Retransmission Time-Out
RTT	Round Trip Time
SACK	Selective Acknowledgment
S-CCPCH	Secondary Common Control Physical Channel
SDU	Service Data Unit
SHO	Soft Handovers
SRNC	Serving RNC
SSTHRESH	slow start threshold
TCP	Transport Control Protocol

TD-CDMA	Time Division Code Division Multiple Access
TF	Transport Format
TR	Transparent Data Transfer Service at the RLC
TTI	Transmission Time Interval
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
UM	Unacknowledged Data Transfer Service at the RLC
USIM	UMTS Subscriber Identity Module
UTRAN	Universal Terrestrial Radio Access Network
VBR	Variable Bit Rate
VLR	Visitor Location Register
WCDMA	Wireless Code Division Multiple Access

Contents

Chapter 1

Introduction

1.1 Motivation

From the very beginning, mobile wireless networks and data networks such as Internet developed separately. With the GPRS (General Packet Radio System), we could see a beginning of unification, where the mobile terminals could access a small part of the existing data services, but the main shortfall of these technologies was the small available bandwidth. This is the reason why the claim of the new 3G (Third Generation) networks is to deliver a high data rate service.

Those who want to speak about the Internet, have to speak about TCP (Transmission Control Protocol). Indeed, TCP accounts for about 95% of the bytes, 90% of the packets and 80% of the flows on the Internet [1]. Therefore, this kind of data traffic will be a main part of the traffic carried by the emerging 3G technologies. TCP is intended for use as a highly reliable host-to-host protocol in packet-switched networks [2]. However, TCP assumes that packets are lost only because of congestion. This was almost true in the case of wired networks, but not anymore in the case of wireless communications where a lot of errors can occur. This is one of the reasons why networks such as UMTS (Universal Mobile Telecommunication System) can use a local ARQ (Automatic Repeat Request) mechanism in order to ensure the reliability of the wireless link. Nevertheless, using a local ARQ in a lower layer than TCP can partly hide the congestion to the TCP congestion control mechanism, and the well known TCP performance could change a lot.

The main goal of this project is to analyze and compare the downlink performance of different TCP protocols over UMTS networks common and dedicated channels, using a Radio Link Control (RLC) in Acknowledged Mode (AM).

1.2 State of the Art

Only a few studies were published about the TCP performance over UMTS networks. The published papers are theoretical or give simulation or emulation results. In the theoretical studies, [3] tries to analyze the impact of different power control methods on the end-to-end TCP performance. Some other analysis concentrate on very specific problems, like [4] which analyzes the importance of the RLC buffer size in UMTS networks. Indeed, a small buffer size leads to unnecessary dropped TCP packets since the buffer is full.

Most of the published simulation studies concentrate on specific UMTS channels with a specific traffic. Some simulations are very simple, like [5] that did not use any ARQ at the link layer, and that only uses the TCP throughput as a quality indicator. The throughput comprises the data sent, the protocol headers and all the retransmitted packets. Therefore, if there are a lot of TCP retransmissions, the throughput is not a good indicator of performance.

The dedicated channel (DCH) was simulated in several papers, but almost never in combination with the common channels. For example, [6] analyses the effect of the TCP advertised window and of some RLC parameters with one user on DCH. One of the main RLC parameter that is tested in this study is the effect of the RLC AM maximum number of retransmissions. The conclusion said it is better not to limit this number. However, the main drawback in this study is only DCH was simulated, and thus there was no congestion problem at the base station. The application used in these simulations was FTP (File Transfer Protocol), that sends as much data as it can. Another similar study analyzed the impact of the status prohibit timer on the TCP throughput, with an infinite number of RLC retransmissions [7]. The status prohibit timer defines how often a RLC bitmap acknowledgment is sent at the receiving RLC. The traffic used in this case was an HTTP traffic, with only one object per page, following a Poisson distribution.

Reference [8] is a lot closer to our concern. The study simulates bursty traffic on a cell with one DCH and a possible switch to DCH from the common channels. The DCH can be allocated to different users following a simple packet scheduling. It analyzes the impact of some switching parameters on the TCP throughput. The wireless link is supposed to be error free. This analysis is the most realistic one, but the traffic tested is not web traffic, and setting an error rate would allow testing the RLC AM parameters.

In all these studies, only one TCP version was used. Most of the time it was Reno or Newreno, but there are no results about the other common TCP versions.

1.3 About this document

This report is divided into the following parts. First, this introduction explains the general motivation of this project. Then, since the settings selected in this study is UMTS, the main characteristics of the UMTS networks will be described, with a detailed description of the different protocols and the different available channels. Then, the main TCP principles will be explained, followed by a description of the main TCP versions.

Afterwards, the project will be described with more details. The problem statement and the approach followed during this project will be clearly defined. The simulation tools and limitations will be described in this part. The next chapter will depict the different implementations that were done during this project.

The three following chapters contains simulations results. The first one shows the congestion problem at the Node B (similarly called Base Station or BS in this document), through the hypothetic case of a FTP (File Transfer Protocol) session over a common channel. The second and the third simulation chapter analyze the performance of different TCP versions using the switch between DCH and Forward Access Channels (FACH) in the downlink. With two different kind of DCH de-allocation mechanisms.

Finally, general conclusions about this project and suggestions for future work are given.

Chapter 2

Overview of UMTS

2.1 Introduction

This chapter is intended to be an overview of the 3G Mobile Communication, the UMTS networks. First of all the main parts of UMTS will be depicted. After that, we will speak about the Wireless Code Division Multiple Access (WCDMA) protocol stack. Then the physical layer and the different channels will be described, and finally some remaining parts of the 3G Mobile Communication basic knowledge will be briefly explained.

2.2 UMTS architecture

According to the 3rd Generation Partnership Project (3GPP) [10], a UMTS network is a "network operated by a single network operator and consisting of UTRAN access networks (WCDMA and/or TD-CDMA), optionally GSM BSS access networks, and UMTS core network". The main parts of the UMTS specific network components, that are shown in Figure 2.1, will be described in the following sections.

2.2.1 User Equipment and External Networks

The user equipment is divided in two parts. The UMTS Subscriber Identity Module (USIM) holds the subscriber identity and store other subscription information. The Mobile Equipment (ME) is the radio terminal used for radio communication over the Uu interface. On the other extremity, we can see the two big existing network types: Circuit Switched (CS) and Packet Switched (PS) networks. The Public Switched Telephone Network (PSTN) is a common example of CS network, while the Internet is PS.

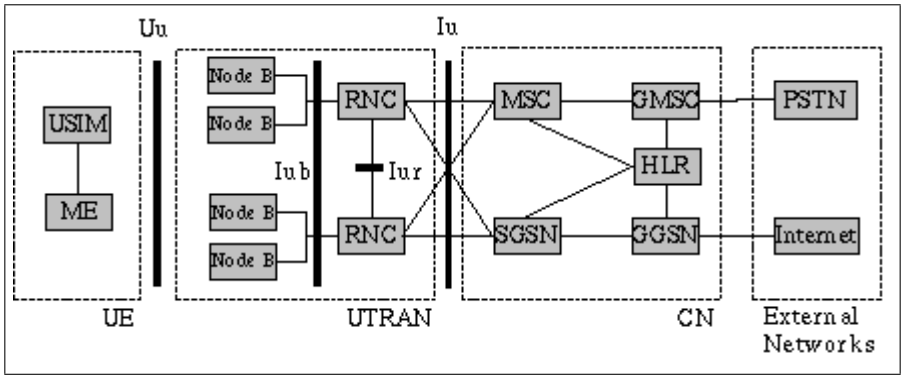

Figure 2.1: UMTS architecture, taken from [12]

2.2.2 UTRAN

The Universal Terrestrial Radio Access Network (UTRAN) is the most important part of the UMTS networks. It handles all radio-related functionality. It is split in two components: Node B and Radio Network Controller (RNC).

Node B

A node B (similarly called Base Station (BS) in this report) is a logical node responsible for radio transmission / reception in one or more cells to/from the User Equipment [10]. The Node B is responsible for coding/spreading on the physical channel. It also participates in Radio Resource Management (RRM). For example, it makes radio measurements for the upper layers, Forward Error Correction (FEC), encoding/decoding of transport channels and Inner loop Power Control [11].

RNC

The RNC is in charge of controlling the use and the integrity of the radio resources [10].

The Radio Network Controller (RNC) controls one or more Node Bs. It is the access point between the Core Network (CN) and UTRAN. It is responsible for the traffic management of the common/shared channels and the Admission control. It also performs Soft Handovers (SHO).

RNC also has a logical role. The RNC controlling a Node B is called Serving RNC (SRNC), and is responsible for the load and control congestion in its own cells. Further, a RNC can be a SRNC or a Drift RNC (DRNC).

The SRNC for one mobile is the RNC that terminate the Iu link for the transport of user data. All traffic from the CN to the mobile will go through this RNC. On the other hand, a DRNC is any other RNC that controls a cell used by the mobile. All communications received by the DRNC of this mobile will be redirected to his SRNC [12].

2.2.3 Core Network (CN)

The CN is responsible for switching, routing procedures and data connections to external networks. It makes the link between the existing external CS and PS networks such as PSTN and Internet. So, the core network is splitted into several CS and PS entities. [12]

MSC/VLR and GMSC

The Mobile Switching Centre (MSC) and the Visitor Location Register (VLR) are the switch and the local database that serve the User Equipment (UE) in its current location for CS services. GMSC is the Gateway MSC.

SGSN and GGSN

The Serving GPRS (General Packet Radio Service) Support Node has the same function than the MSC/VLR, but for PS services. GGSN is the Gateway GPRS Support Node.

HLR

The Home Location Register (HLR) is a database located in the user's home system that contains the master copy of all information about the user's service profile. For example, it contain information about roaming areas, allowed services, authentication keys, etc.. It is created when a new user subscribes to the system and remains stored as long as the subscription is active.

2.2.4 Interfaces

An interface defines the connection between different entities. The main interfaces are well-specified by the 3GPP. With these, the equipment on different ends of the interface can be acquired from different manufacturers. It is necessarily to enable competition between them in order to decrease the prices [11]. The most important ones are:

Iu

This interface connects the CN to the UTRAN. It can be divided in two parts: Iu-CS for the CS domain and Iu-PS for the PS domain.

Iub

This interface is situated between the RNC and the Node B in the UTRAN. The tasks that Node B and RNC have to perform together are so complex that a proprietary solution is the most likely one. For example, the traffic management of common/shared/dedicated channels and system information management are performed on this link.

Iur

The Iur interface connects two RNCs. It is used for example for the DRNC-SRNC connectivity.

2.3 WCDMA air interface protocol stack

To help the transparency between the different Mobile Communication Generations, the concepts of Access Stratum (AS) and Non Access Stratum (NAS) were defined. The AS is a functional entity that includes radio access protocol between the UE and the UTRAN. The NAS includes CN protocols between the UE and the CN itself [11]. Thus, in theory, NAS protocols are independent of the underlying radio access technology and can be reused from older Global System for Mobile communication (GSM) networks. This section will focus on the AS protocols specific to UMTS, that are typically implemented in the UE and in the RNC. Note that the Node Bs are not aware of Radio Link Control (RLC) and upper protocols. This section will describe the UMTS AS protocols, shown in Figure 2.2.

2.3.1 Radio Resource Control

The Radio Resource Control (RRC) is a sub layer of radio interface Layer 3 existing in the control plane only which provides information transfer service to the non-access stratum. The RRC is responsible for controlling the configuration of radio interface Layers 1 and 2 [10].

RRC functions and signaling procedures

The major part of the control signaling between UE and UTRAN is RRC messages [11]. They carry all the parameters required to set up, modify and release layer 2 and 1 protocol entities [12]. Among other things, RRC

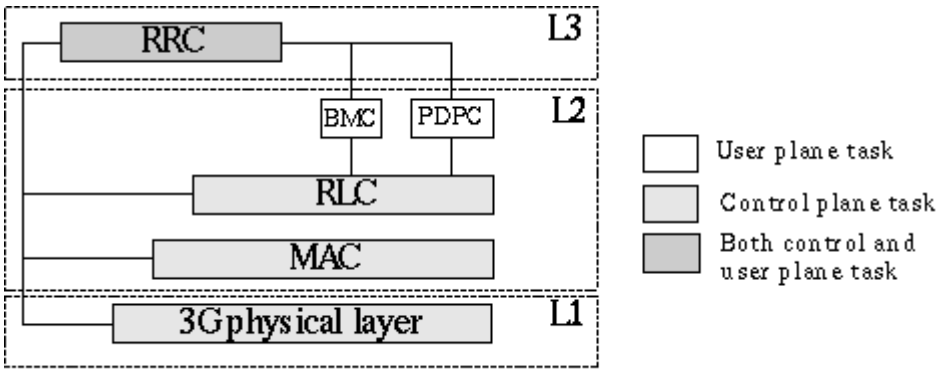

Figure 2.2: Protocol stack in the UTRAN (only AS), taken from [11]

is responsible for outer-loop power control, security mode control (through ciphering and integrity protection), Handovers (HO), reception of paging messages, measurements, etc.. RRC is also responsible for the establishment, reconfiguration and release of radio bearers.

RRC Service States

The two basic modes of a UE are idle mode and connected mode [12]. In the idle mode, the UE is "camping on a cell". However, it is still able to receive signaling information such as paging. The UE will stay in this state until a RRC connection is established. The connected mode can be further divided into service states.

1. *Cell_DCH*

   In this state, a dedicated channel (DCH) is allocated for the UE, and the UE is known by its SRNC.

2. *Cell_FACH*

   Here, common channels are used to send signaling and small or bursty amount of data. Random Access Channel (RACH, for the uplink direction) and Forward Access Channel (FACH, for the downlink direction) are used. Because the UE has to continuously monitor the FACH, it consumes power. Thus, if the channel is no longer used, the RRC switches to the *Cell_PCH* state.

3. *Cell_PCH*

   The RRC connection still exists, but the UE can only receive broadcast messages and messages from the Paging Indicator Channel (PICH). The UE cannot send anything.

4. *URA_PCH*

   This state is close to the previous one, except that a cell-update procedure is only triggered if the UTRAN registration area changes.

What makes UMTS so different from the other mobile telecommunication systems is that it allows adapting and changing bearers characteristics following the application needs. This allows link usage optimization and better end-to-end transport services. For example, real-time bit rate, Constant Bit Rate (CBR), Variable Bit Rate (VBR) traffic have different requirements in term of delay variation, Bit Error Ratio (BER) and data rate. Thus, a negotiation routine exists that takes care of the current network congestion (through the admission control) and the required quality of service. Figure 2.3 shows this routine and the possible RRC state changes.

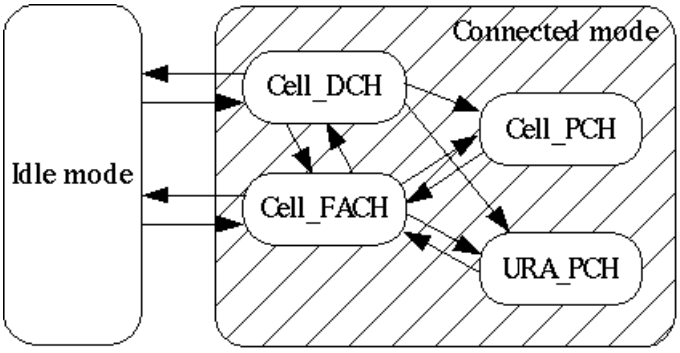

Figure 2.3: UE modes and RRC states in connected mode (taken from [12])

The main parameters used to switch from a *Cell_FACH* state to a *Cell_DCH* are the upswitch RLC buffer threshold and the downswitch inactivity timer. The upswitch buffer threshold is the RLC buffer size above which a DCH should be allocated, and the downswitch inactivity timer is the time an inactive UE stays in the *Cell_DCH* state before a downswitch to the *Cell_FACH* state can be triggered.

2.3.2 Layer 2

BMC/PDCP

The Packet Data Converge Protocol (PDCP) must hide the particularities of each protocol from the UTRAN [11]. The PDPC functions includes, among other things, header compression and decompression of IP data streams and

the transfer of user data.

The Broadcast/Multicast Control (BMC) only handles downlink broadcast/multicast transmission. It implements the Cell Broadcast Service (CBS) messages that are broadcast to every user in a cell.

RLC

RLC is a sub layer of radio interface layer 2 providing transparent, unacknowledged and acknowledged data transfer service [10]. A wireless physical channel is characterized by a high error rate. Normally, errors are corrected by the transport layer if needed (for example with TCP), but because errors occur often, it can be faster to do retransmissions in a lower layer. That is the goal of this sub layer. Note that in order to do this, the RLC performs segmentation and reassembly of higher-layer Service Data Units (SDU) into/from smaller RLC payload units (called Protocol Data Unit, PDU).

The following services are provided to upper layers [11] [12]:

1. Transparent Data Transfer Service (TR)

   In this mode, the only operation that can be done by the RLC is segmentation and reassembly, but even this functionality is very limited, since the RLC doesn't add a protocol header. If this is used, it has to be negotiated in the radio bearer set up procedure. Erroneous packets are dropped or marked.

2. Unacknowledged Data Transfer Service (UM)

   In this mode, data delivery is not guaranteed. Segmentation and reassembly is done by adding a header to RLC PDUs. Those PDUs also contains sequence numbers so that the integrity of higher layer PDUs can be checked. The packets are encrypted.

3. Acknowledged Data Transfer Service (AM)

   Here, an Automatic Repeat Request (ARQ) mechanism is used to reduce the error rate for higher layers. The quality vs delay performance of the RLC can be configured by the RRC, by changing the maximum number of retransmissions (maxDAT). Moreover, the RLC can be configured for in-sequence or out-of-sequence delivery. The packets are encrypted.

MAC

The main function of the Medium Access Control (MAC) sub layer is the mapping between the logical channels recognized by the RLC and the transport channels used by the physical layer. New data is sent each Transmission Time Interval (TTI) to the physical layer.

The logical channels specify the kind of data flowing over the network, though the transport channels defines how the data is transferred. To do this, the MAC protocol has to select the appropriate Transport Format (TF).

Among others, the MAC layer is also responsible for:

- Some traffic volume monitoring, that will be used by the RRC for triggering reconfiguration of Radio Bearers and/or Transport channel
- Ciphering if the RLC is in TR
- Multiplexing/demultiplexing of higher PDUs into/from transport block
- Dynamic transport channel type switching (the switching decision between common and dedicated channel come from the RRC.

The MAC layer consists of three logical entities (see Figure 2.4). There is only one MAC-b and one MAC-c/sh in each Node B and in each UE. However, there is one MAC-d entity in each UE and one MAC-d entity for each UE in the SRNC. The possible mappings between logical channels and transport channels is shown in Figure 2.5, while table 2.1 and 2.2 briefly describe the logical and transport channels.

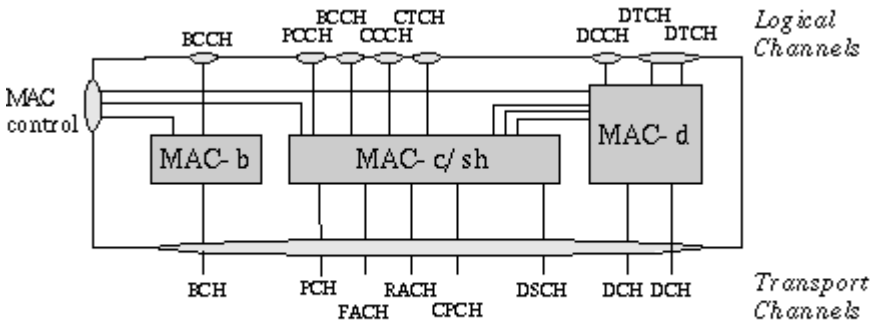

Figure 2.4: MAC layer architecture, taken from [12]

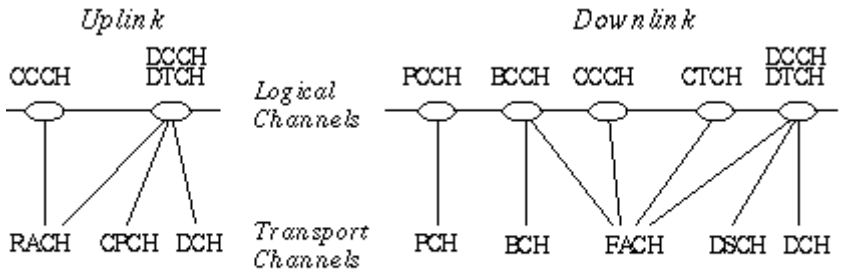

Figure 2.5: Mapping between logical channels and transport channels, taken from [12]

Logical channels		Direction
BCCH	Broadcast Control Channel	DL
PCCH	Paging Control Channel	DL
DCCH	Dedicated Control Channel	UL/DL
CCCH	Common Control Channel	UL/DL
DTCH	Dedicated Traffic Channel	UL/DL
CTCH	Common Traffic Channel	UL/DL

Table 2.1: Logical Channels

Transport channels		Direction
BCH	Broadcast Channel	DL
PCH	Paging Channel	DL
RACH	Random Access Channel	UL
CPCH	Common Packet Channel	UL
FACH	Forward Access Channel	DL
DSCH	Downlink Shared Channel	DL
DCH	Dedicated Channel	UL or DL

Table 2.2: Transport Channels

Note that RACH is only for limited-size data fields, while FACH is intended for bursty data traffic.

DCH is the only dedicated channel. If the data traffic over the DCH suffers from some peaks, it can be associated with a DSCH to help sending these peaks. This system allows not to give a too large bandwidth to the DCH if the peaks are not frequent.

There is a last transport channel that can be added: the High Speed Downlink Shared Channel (HS-DSCH). It is optimized for very high speed data transfer, and is always associated with a DCH.

2.3.3 Physical layer

The physical layer mainly maps transport channels into the right physical channel. Figure 2.6 shows the mapping between the transport channels and the physical channels, and Table 2.3 briefly describe the physical channels. Only channels that are relevant in term of services for the upper layer are showed here.

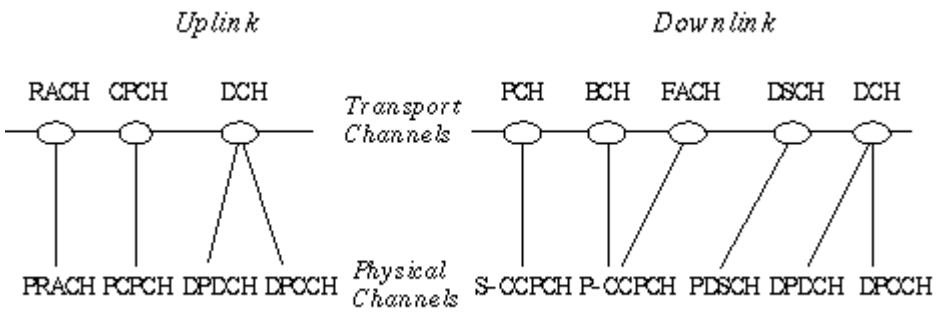

Figure 2.6: Mapping between transport channels and physical channels, adapted from [11]

The UMTS physical layer uses WCDMA as its air interface. Its description will be very brief, and is only present in order to make this overview complete.

Figure 2.7 shows how Code Division Multiple Access (CDMA) works for one user. It is the same when multiple users are sending data. Among other things, CDMA is used in order to send more than one data flow in the same time, to decrease the multiple access interference from many system users

Physical channels		Direction
PRACH	Physical Random Access Channel	UL
PCPCH	Physical Common Packet Channel	UL
DPDCH	Dedicated Physical Data Channel	UL and DL
DPCCH	Dedicated Physical Control Channel	UL and DL
P-CCPCH	Primary Common Control Physical Channel	DL
S-CCPCH	Secondary Common Control Physical Channel	DL
PDSCH	Physical downlink Shared Channel	DL

Table 2.3: Physical Channels

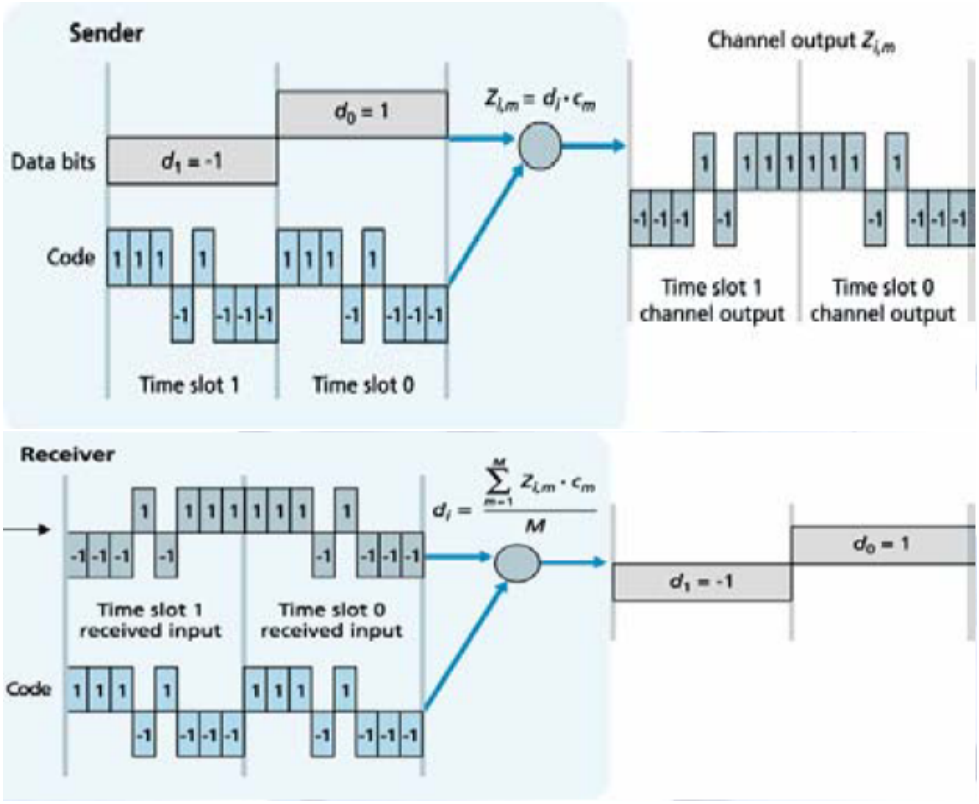

Figure 2.7: Spreading-despreading in CDMA (single user), taken from [15]

and to provide variable bit rates [12].

The spreading codes are chosen through the channelization and the scrambling mechanisms.

1. Channelization

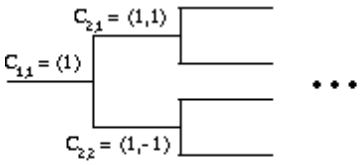

Figure 2.8: Beginning of the channelization code tree, taken from [12]

The spreading/channelization codes of the Universal Terrestrial Radio Access (UTRA) are based on the Orthogonal Variable Spreading Factor (OVSF) technique. The codes are picked from the channelization tree (see Figure 2.8). This allow UTRA physical layer to give a higher bandwidth to a connection, by giving it a higher channelization code. However, when a code is allocated to an user, no other code from an underlying branch can be used (in order to respect the orthogonality).

2. Scrambling Scrambling is needed to separate terminals or base stations from each other, and is used in top of spreading (see Figure 2.9).

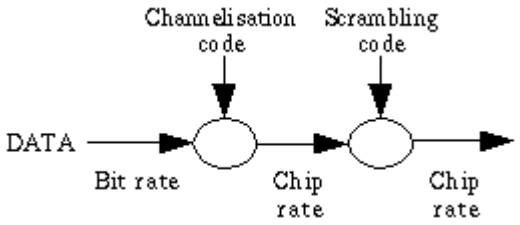

Figure 2.9: Relationship between spreading and scrambling, taken from [12]

2.4 Procedures

This section summarize two of the main UMTS procedures: the handovers (HO) and the power control.

2.4.1 Handovers

A HO is the transfer of a user's connection from one radio channel to another (can be the same or different cell) [10]. For example, a UE can maintain its connection in cellular networks when it moves from one cell area to another one [11]. The biggest problem with HO is the long delays.

There are three kinds of HO:

Soft Handovers

A Soft Handovers (SHO) takes place when a UE receives data from more than one base station at the same time [11] [12]. This requires the use of two separate codes in the downlink direction, so that the UE can distinguish the signals.

Softer Handovers

A Softer HO is a special type of SHO where only one of the possible base station is sending data to the UE [11].

Hard Handovers

A Hard Handovers (HHO) is also known as an inter-frequency HO [11]. The problem is making the required measurements on the new channel. Indeed, a CDMA handset is transmitting continuously in connected mode, so there are no spare slots available to make measurements in other frequencies. To resolve this problem, the compressed mode can be used. That means that not all time slots in downlink are used for data transmissions. When the required measurements are done, the UE stops transmitting on the old frequency and begins using the new one.

Intersystem Handovers

Intersystem HOs are HOs between two different Radio Access Technologies (RAT) (for example, between a GSM/GPRS and an UMTS). A lot of measurements have to be done before such a HO can take place. In order to do this, the compressed mode can be used [11].

2.4.2 Power control

The main purpose of the Power Control (PC) is to minimize the interference in the system, in order to keep the Bit Error Rate (BER) as constant as possible. In the uplink direction, all signals should arrive in the Node B with the same signal power, whether the UE is close or not. In the downlink, signals sent by other Node Bs are not orthogonal and thus increase the interference

level. That is one of the reasons why signals should be transmitted with the lowest possible power level [12].

There are two different ways to operate this power control:

Open-loop power control

The transmitting entity estimates the required power level by itself from the received signal.

Closed-loop power control

This power control is based on the explicit power control commands received from the peer entity. This power control can be further divided into inner-loop power control and outer-loop power control.

Chapter 3

Transmission Control Protocol overview

This chapter intends to make a little summary of the common Transmission Control Protocol (TCP) mechanisms and the way they are working. Moreover, although the most common TCP principles never changed, a lot of improvements were proposed from the original TCP. Each version has its own strengths and weaknesses. First, the most important characteristics of TCP will be described. Then, the different ways for TCP to react to congestion will be detailed. Finally, there will be a brief explanation of the main TCP versions.

3.1 TCP basics

3.1.1 Connection oriented and reliable data transfer

The main goal of TCP is to provide a reliable host-to-host data transfer on top of a non-reliable layer, usually the Internet Protocol (IP).

Reliable data transfer

TCP ensures that the data transfer is reliable because [13]:

- The application data is broken into what TCP considers the best sized segments to send, identified by a sequence number.
- When TCP sends a segment, it starts a timer known as the Retransmission Time-Out (RTO), waiting for the other end to acknowledge reception of the segment (by sending an acknowledgement (ACK)). If an ACK is not received before the RTO expires, the segment is retransmitted.

- TCP maintains a checksum on its header and data. If the receiver has an invalid checksum compared with the data, it simply discards the packet. Normally, the sender will retransmit the corrupt segment as no corresponding ACK will be received before the RTO expires.
- A TCP receiver discards duplicate data.
- TCP will reorder packets received out-of-order.
- TCP also provides flow control. A receiving TCP only allows the other end to send as much data as the receiver has buffers for.

TCP header

The TCP sender and receiver must share some information (like acknowledgments and receiver's buffer size). This information is sent within a header appended to each TCP segment. The standard size of this header is 20 bytes, but some protocols use 20 more bytes for TCP options (see Figure 3.1).

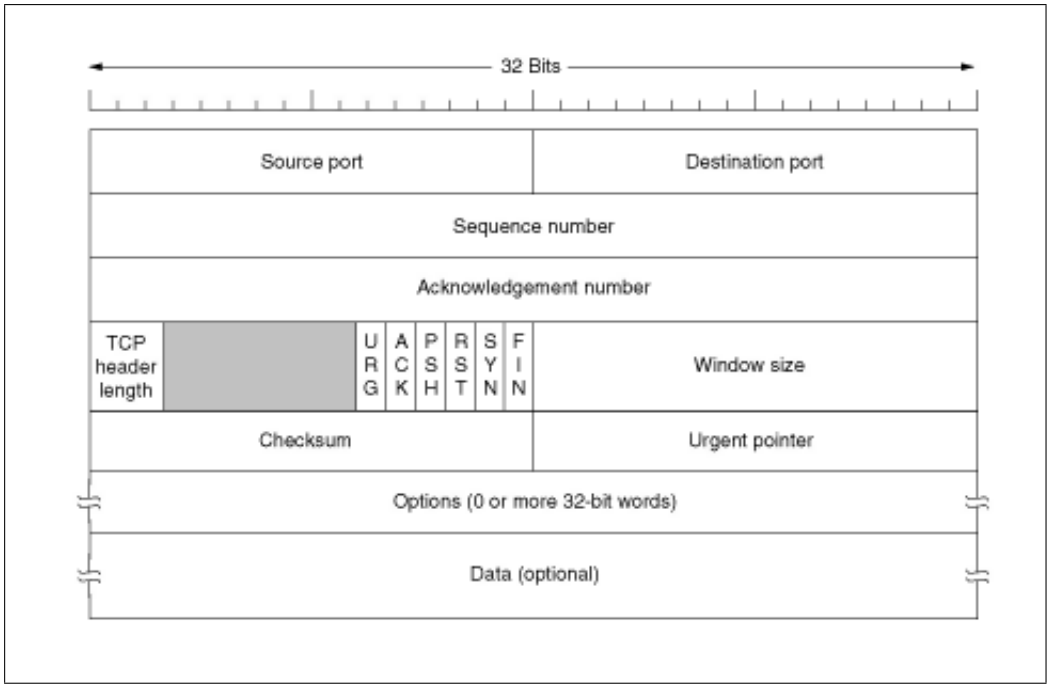

Figure 3.1: TCP header

In order to be reliable, the two hosts using TCP must be aware of the connection, and be synchronized with each other. From this we can say that

TCP is a connection-oriented protocol, since it needs a connection establishment procedure to synchronize the sender and receiver of the data, and a connection termination procedure to inform the hosts that there is no data to transmit anymore.

Connection establishment and termination

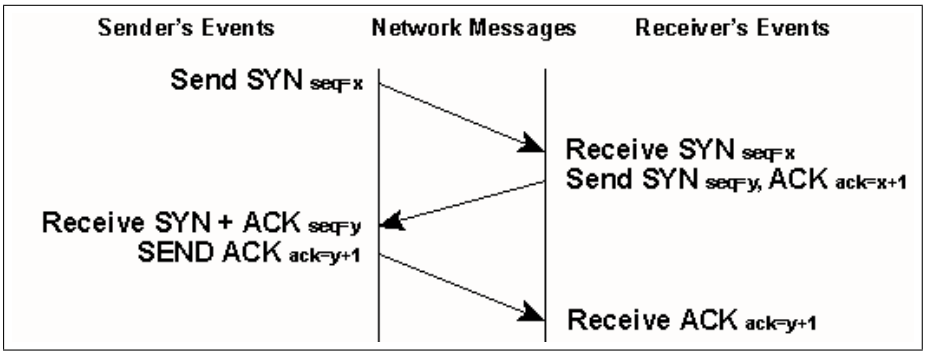

Figure 3.2: TCP handshake procedure

The connection establishment procedure is called "handshake", and the segments sent at this time only contains the header, usually 40 bytes (a 20 bytes TCP header and a 20 bytes IP header).

1. The requesting end (called a client) sends a SYN segment with no data to the other end (called the server), meaning that the client wants to initiate a connection. This segment must contain the Initial Sequence Number (ISN) of the segments, and the Maximum Segment Size (MSS) that will be used for this connection.
2. The server replies with its own SYN segment containing a piggybacked ACK for the client's SYN.
3. The client acknowledges the server's SYN, and the connection is established for the two ends.

While TCP uses only 3 segments to initiate the connection, it needs 4 segments to terminate it. This comes from the TCP's half-close state (see Figure 3.3).

1. At the beginning, the end who wants to terminate the connection (for example the client) sends a FIN segment to the other end.

2. When receiving a FIN segment, the server has to acknowledge it, but can still send data to the client. The TCP connection is in half-close state.

3. When the server decides to close the connection, it sends a FIN segment

4. The client acknowledges the FIN segment. At this time, the connection is completely closed.

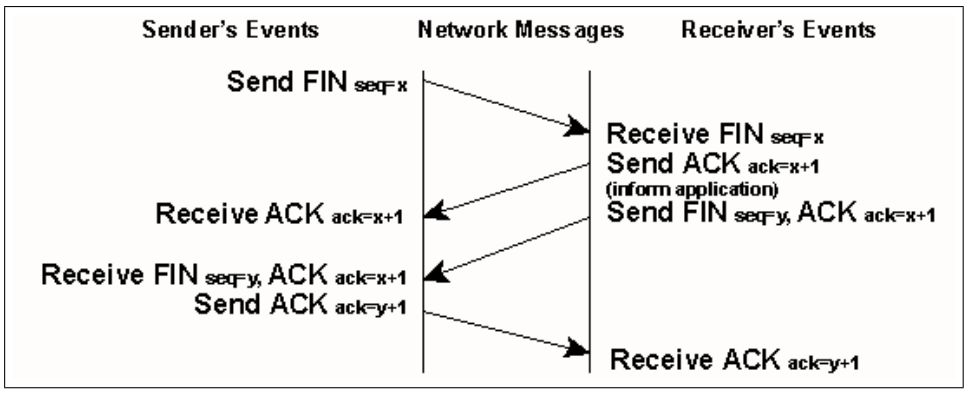

Figure 3.3: TCP termination procedure

3.1.2 Flow control

Practically, there are two different ways to send data. The first way is called "Stop And Wait". This mechanism is very inefficient, because the sender only sends one segment at a time, and waits for the receiver to acknowledge it. The second way is called "sliding window", and permit to send as much segments as the receiving end can handle, before having to wait for acknowledgment [19]. TCP uses the sliding window mechanism.

In this mechanism, a certain number of segments (called a window of segments) are transmitted at once. Each segment has a sequence number appended to it, and the receiver can acknowledge more than one segment at a time by acknowledging the highest one received, meaning that all the previous segments were successfully sent.

This flow control needs some information to be shared between the sender and the receiver. For this, a field called the advertised window (AWND) in the TCP header (Figure 3.1) is used to inform the sender of the receiving buffer size. The sending window is limited by the AWND, so a fast sender

will not overwhelm a slow receiver.

3.1.3 Congestion control

If the sender is only limited by the AWND, a lot of packets can be dropped because of a full buffer in an intermediate router. Therefore the sending window must be limited not only by the receiver buffer, but also by the network capacity. The window size resulting from the congestion control is called the congestion window (CWND). The sending window is taken as being the minimum of the AWND and the CWND.

If a packet is lost, TCP retransmits it (and all the following packets) through its Automatic Repeat Request (ARQ) mechanism. This kind of ARQ is called Go-Back-N.

Slow Start

The way in which TCP data transmission operates during the start of a connection is known as slow start [13].

The slow start algorithm avoids the congestion problem by observing that the rate at which new packets should be injected into the network is the rate at which the acknowledgments are returned by the other end.

The sender starts by transmitting one segment and waiting for its ACK. Afterwards, CWND is increased by one segment each time an ACK is received.

The main drawback to slow start is the large amount of time that is required during start up. If the data that is being sent is very small, the bandwidth efficiency will be reduced considerably [19].

Congestion Avoidance

The slow start increases the CWND exponentially. At some point during the connection a bottleneck in the network will be congested and starts discarding packets [13]. Therefore, above a certain threshold, an exponential increase of CWND seems inappropriate to find the right CWND value.

The relation between slow start and congestion avoidance is done through a variable called Slow Start Threshold (SSTHRESH). If CWND is below SSTHRESH, the TCP sender is in slow start, otherwise it is in congestion

avoidance, meaning that CWND is increased only by 1/CWND each time an ACK is received. This is an additive increase (Figure 3.4).

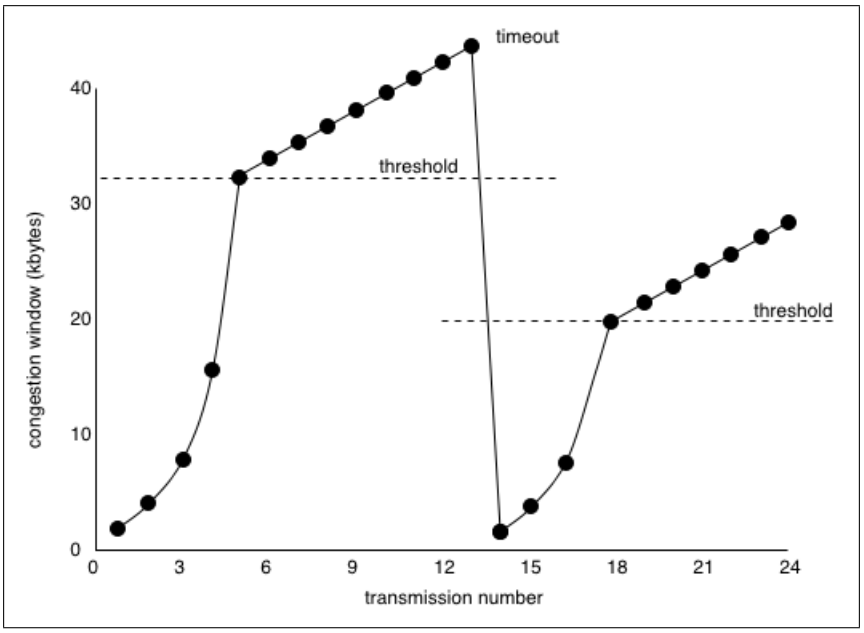

Figure 3.4: Slow Start and Congestion Avoidance

TCP assumes that almost all packet losses are due to congestion somewhere in the network. Therefore it is necessary to reduce the amount of segments to be sent if a packet loss is detected (indicated by a RTO or the reception of duplicate ACKs).

When congestion occurs, SSTHRESH is set to [13]:
$SSTHRESH = Max(Min(CWND, AWND), 2)$

The TCP behavior is different if the congestion was detected through an RTO or some duplicate ACKs.

TCP Retransmission Time-Out

When data segments are not received, the RTO expires, TCP retransmits the segment and goes back to slow start. However, the RTO cannot be fixed, since the time between when a packet is sent and its ACK arrives (known as the Round Trip Time (RTT)) can change a lot depending on the network. The RTO is calculated in the following way, as proposed in the original TCP specifications [20]:

$$RTO = SRTT + 4 * Deviation$$
$$SRTT = \frac{7}{8} * SRTT + \frac{1}{8} * SampleRTT$$
$$Deviation = \frac{3}{4} * Deviation + \frac{1}{4} * |SampleRTT - SRTT|$$

where SampleRTT is the last calculated RTT value.

It can be seen that the RTO is dependent on the last RTT sample, and in some part from the past RTTs. When a timeout occurs, the RTO is doubled, with a maximum of 64 seconds [13].

Duplicate ACKs

When 3 duplicate ACKs are received by the TCP sender, the following TCP mechanisms can take place.

1. Fast Retransmit

   When a RTO occurs, it means that almost no packets could go through the network because of congestion. However, if the TCP sender receives duplicate ACKs, it means that a packet was lost, but that some others reached the receiver. In this case, TCP will retransmit the concerned packets without waiting for the RTO [14] [13].

2. Fast Recovery

   When the third ACK duplicate is received, TCP performs congestion avoidance instead of slow start, since it does not want to reduce the flow abruptly by going into slow start. SSTHRESH is set to one-half the current window, but CWND will be set at SSTHRESH plus three (because we received three duplicate ACKs) [19] [13]. Each time another duplicate ACK arrives, CWND is incremented by one. When an ACK that acknowledges new data arrives, TCP goes to congestion avoidance.

3.2 TCP versions

3.2.1 TCP Tahoe

These are the main features of TCP Tahoe [20]:

- Slow Start
- Congestion avoidance
- Fast retransmit

The main problem of Tahoe is that even if some packets can still flow through the network, it will perform slow start if a packet loss is detected, abruptly reducing the flow.

3.2.2 TCP Reno

These are the main features of TCP Reno [20] [16]:

- Slow Start
- Congestion avoidance
- Fast retransmit
- Fast Recovery

TCP Reno extends Tahoe by introducing the Fast Recovery mechanism. This solves the main Tahoe problem, but can lead to a stall. Indeed, TCP Reno goes out of Fast Recovery when it receives a new partial ACK (those which represent new ACKs but do not represent an ACK for all outstanding data). That means that if a lot of segments from the same window are lost, TCP Reno is pulled out of Fast Recovery too soon, and it may stall since no new packets can be sent. Indeed, in Congestion Avoidance, CWND is increased by 1/CWND each time a new ACK is received, though in Fast Recovery CWND is increased by 1.

3.2.3 TCP Newreno

These are the main features of TCP New Reno [20] [16]:

- Slow Start
- Congestion avoidance
- Fast retransmit
- Enhanced Fast Recovery

TCP New Reno has the same behavior than TCP Reno, but tries to avoid the TCP Reno stall problem by ignoring the new partial ACKs in Fast Recovery. TCP New Reno will only be pulled out of Fast Recovery when an ACK with a higher value than the highest seen so far is received.

3.2.4 TCP Vegas

TCP Vegas introduces some innovations. The sender stores the current value of the system clock for each segment it sends. Therefore it is able to know the exact RTT for each sent packet.

These are the main features of TCP Vegas [21]:

- Fast retransmit
- Fast Recovery
- *New Retransmission mechanism:* If the RTT of a duplicate ACK is greater than a threshold, the segment is retransmitted without waiting for the classic RTO nor for 3 duplicate ACKs.
- *New Congestion Control mechanism:* It depends on the difference between the calculated throughput and the value it would achieve if the network was not congested. When that difference is smaller than a boundary, the window is increased linearly. When it is larger than another higher boundary, it is decreased linearly. The throughput of an uncongested network is defined as the window size in bytes, divided by the BaseRTT, which is the value of the RTT in an uncongested network. The RTT in an uncongested network is calculated as the RTT of the last packet that was sent alone.
- *New Slow Start mechanism:* CWND is doubled every time the RTT changes instead of every RTT. To exit the Slow Start phase, a boundary is set on the difference between the current RTT and the last RTT instead of a boundary on the CWND size.

3.2.5 TCP Sack

These are the main features of TCP Sack [22]:

- Slow Start
- Congestion avoidance
- Fast retransmit
- Modified Fast Recovery (see below)
- Sack mechanism (see below)

Sack was implemented in order to stay as close as possible to TCP Reno, while adding a full Selective Repeat and Selective Acknowledgement mechanism. TCP Sack uses the option field in the TCP header to store sets of properly received sequence numbers. The main difference between TCP Sack and TCP Reno is in the behavior when multiple packets are dropped from one window of data [22].

During Fast Recovery, Sack maintains a variable called *pipe*, that represents the estimated number of packets outstanding on the link. The sender only sends new or retransmitted data when the value of *pipe* is less than the congestion window (CWND). The variable *pipe* is incremented each time the sender sends a packet, and is decremented when the sender receives a duplicate ACK with a SACK option reporting that new data has been correctly received.

The sender maintains a *scoreboard* that remembers acknowledgments from previous SACK options. When the sender is allowed to send a packet, it sends the next packet known as missing at the receiver if such a packet exists, otherwise it sends a new packet. When a retransmitted packet is lost, Sack detects it through a classic RTO and then goes into Slow Start. The sender only goes out Fast Recovery when an ACK is received acknowledging all data that was outstanding when Fast Recovery was entered. Because of this, we can say that Sack is closer to New Reno than to Reno, since partial ACKs do not pull the sender out of Fast Recovery.

3.2.6 TCP Fack

TCP Fack is an extension of TCP Sack and has the same functionalities. However, it improved some parts of the TCP Sack protocol:

- *More precise estimation of outstanding data:* The information of the SACK option is being used to better estimate the amount of data in transit [23].
  data in transit = $snd.nxt - snd.fack + retransmitted$
  where snd.nxt is the first byte of data not sent, snd.fack is the highest sequence number known to have been received plus one and retransmitted is the number of retransmitted segments.

- *Data smoothing:* A better way of halving the window when congestion is detected. When the CWND is immediately halved, the sender stops transmitting for a while and then resumes when enough data has left the network. This unequal distribution of segments over one RTT can be avoided when the window is gradually decreased [23].

- *Slow start and congestion control:* The window should be halved according to the multiplicative decrease of the correct CWND when congestion occurs. Given the fact that the sender identifies congestion at least one RTT after it happened, if during this RTT it was in Slow-Start mode, then the current CWND will be almost double then the CWND when congestion occurred. Therefore, in this case, the CWND is first halved to estimate the correct CWND that should be further decreased [24].

Chapter 4

Problem Statement

4.1 Problem definition

As said in the introduction, the main goal of this project is to analyze and compare the performance of different TCP protocols over UMTS networks common and dedicated channels. As the downlink capacity is expected to be more important than the uplink capacity because of the asymmetric traffic type [12], only the downlink performances are analyzed.

Since the goal of this project is to analyze TCP performance, the Acknowledged Mode of the RLC was chosen. The tested RLC parameters are the maximum number of RLC retransmissions, the upswitch buffer threshold at the BS (in the downlink), and the downswitch delay.

4.2 Approach

A discrete event simulation approach has been chosen for this project. Each packet generates an event when it is handled by the simulator (see Section 4.3). The main advantage of this approach is that it is really faster than an emulation or a real testing, and that it is often easier to implement than a real system.

Figure 4.1 shows the general topology that was used for the simulations. The continuous lines represent wired links while the wireless links are represented by the dotted lines. There are n nodes connected to the GGSN and n UEs. Each wired node is a TCP sender and each UE is a TCP receiver.

Web traffic was chosen for the main simulations since it is one of the most important data traffic that will be transported on common/dedicated channels. The main quality indicators that were considered for such a traffic

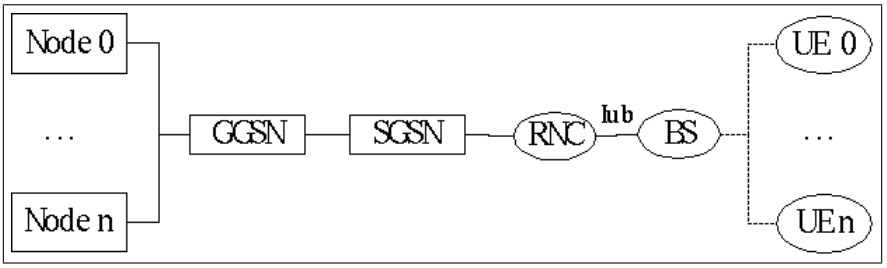

Figure 4.1: General simulation topology

are the TCP goodput (data sent, without headers and retransmissions, per given time), the number of successful page calls (a user's request for information [9]) and the average time necessary to complete a packet call. Some other simulations (including the simulations on the common channels only) used an FTP traffic.

The errors occurs between the BS and the UE, thus the "discard unit" is the RLC PDU (Protocol Data Unit). Therefore, it is a Block Error Rate (BLER) at the RLC level. The error rate is supposed constant. Two different error rates will be used during the simulations. When the traffic was a FTP session, a standard error rate was used.

A multistate error rate was used for the main simulations. It uses a two-states Gilbert model to show the correlation between packet losses. In the used Gilbert model, there is an error-free state (0) and an error state (1). Figure 4.2 shows the Gilbert model state transition diagram.

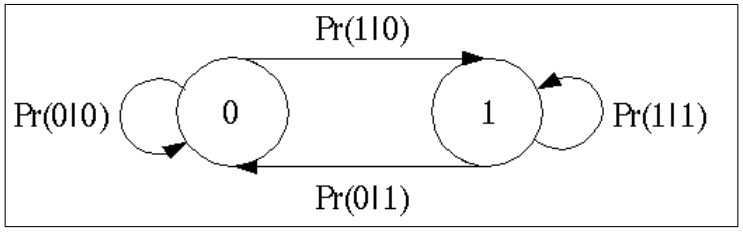

Figure 4.2: Gilbert model state transition diagram, taken from [25]

The corresponding error rate, Pr(error), is calculated as follow [26]:

$$Pr(error) = \frac{Pr(1|0)}{Pr(1|0)+Pr(0|1)}$$

4.3 Tools

4.3.1 Network Simulator 2

The chosen discrete event simulator is the Network Simulator (NS2) [16]. NS2 is an open-source simulator widely used in the academic community. Therefore a lot of people are working on this project and there is a wide variety of add-ons. NS2 is implemented in two parts: a TCL interpreter to make the simulation scripting easier and a C++ implementation to have faster simulations. Figure 4.3 shows a simplified user's view of NS.

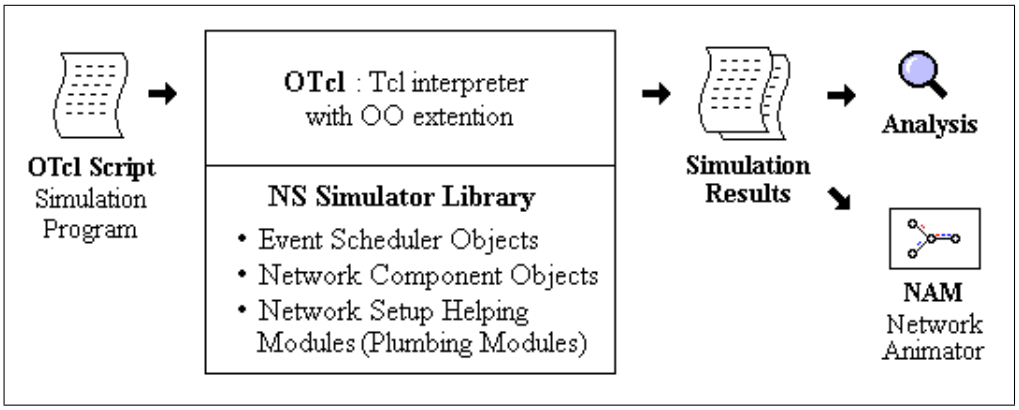

Figure 4.3: Simplified User's view of NS, taken from [18]

A lot of standard network features and protocols are implemented in NS2. It is not the goal of this section to summarize all the existing functionalities of NS2, but just to mention some of them. All the TCP versions described in Chapter 3 are implemented. Some applications like FTP already exists too. The lower layers (MAC and physical) are simulated as well for wired networks.

Some parts of TCP are only partially implemented in NS2. The use of the different TCP versions in NS2 is limited to one-way TCP connection, with a partial TCP handshake. A synchronization packet is sent by the client and acknowledged by the server, but no synchronization packet is sent from the server to the client. The connection termination is not implemented at all.

There is no advertised window (AWND) mechanism in NS2. The advertised window is supposed to be always equal to 20 packets by default.

4.3.2 Eurane

EURANE (Enhanced UMTS Radio Access Network Extensions for NS2) is the NS2 UMTS extension developed within the framework of the IST SEACORN project[17] that was used during this project. This extension limits the simulations to one cell, and as such no handover is implemented. EURANE includes three additional nodes (the RNC, BS and UE, see Chapter 2 for their meaning), whose functionality allows for the support of FACH, RACH, DCH and HS-DSCH. The main functionality additions to NS2 are the RLC in AM and UM, as well as some MAC support for the new transport channels. In sequence delivery is used and no power control is implemented.

Chapter 5

Implementations

This chapter will describe the implementations that were made in order to make this project possible. The two first implementations, that are specific to a UMTS environment, were made on top of EURANE. The WEB traffic generator and the trace analyser are more general, and are made on top of the unpatched NS2.

5.1 Maximum RLC retransmission limit

The original EURANE implementation does not limit the number of RLC retransmissions (maxDAT). This is an important parameter, since one of the worst problems on the common channels is the congestion at the BS. Therefore, infinite retransmissions at the RLC can cause TCP to almost never being aware of the congestion, since there are no packet losses, and thus it would never adapt its sending rate (see Chapter 6).

When a PDU is retransmitted too many times, the corresponding SDU has to be discarded. However, a single SDU is often segmented in a lot of smaller PDUs, and all the associated PDUs have to be discarded too. There is no problem to discard them on the sender side, but a specific message has to be sent to the receiver so it can discard the PDUs belonging to the concerned SDU. This message is sent through a status message, with a MRW (Move Receiving Window) indication [29]. This MRW contains the sequence number (SN) of the last PDU of the discarded SDU, so the receiver can discard all the not yet successfully received SDUs that have a PDU SN $< MRW$ and move the receiving window accordingly.

The 3GPP specification allows to send several MRW in a single status message [29]. However, in the modified EURANE implementation, only one MRW is sent at the same time. The RLC sender tries to piggyback the

status inside the next packet to be sent, and if it is not possible it sends it immediately with padding.

The MRW message can be lost due to congestion at the base station or because of a transport channel error. Therefore it is necessary that the receiver acknowledges the status messages containing MRWs. The 3GPP specification allows the use of a MRW ACK status message. However, this kind of message was not implemented. It is possible to know if the MRW message was successfully received when the sender receives the next bitmap acknowledgment. If $MRW < FSN$, where FSN is the first sequence number of the received bitmap, then the sender knows that the concerned MRW was successfully received. After a certain delay (500ms by default in the simulator), if a MRW is not acknowledged, the sender sends the MRW message again.

5.2 DCH allocation

The second EURANE modification is a limitation of the total cell bandwidth allowed for the DCH allocation. The implemented packet scheduling is quite simple compared to the 3GPP specification. There is no dynamic bandwidth allocation and the same bandwidth is allocated to all DCH. The connection uses the common channels (FACH and RACH) until the RLC buffer reaches a threshold [12]. When this threshold is reached, and if there is enough free bandwidth in the cell, a switch to a DCH is triggered. If not, the user simply keeps using the common channels. Please note that each RRC change state, as the allocation and deallocation of a DCH, takes a certain time. It is not specified if the user can continue to use the common channels or not when allocating a DCH. The examples in [12] consider that the user cannot use the common channels while allocating a DCH. This is the approach that was considered in this report.

Figure 5.1 shows the RLC buffer occupancy during a TCP file download. The transfer is divided in 4 periods:

1. During the first period, the user is on FACH/RACH and the buffer occupancy is below the threshold. Then, the file transfer starts and the buffer occupancy becomes quickly bigger than the threshold.

2. During the second period, a DCH has been triggered if there was enough free bandwidth on the cell, but it needs a certain time to be allocated. No data is sent during this period.

3. During the third period, a DCH is allocated to the connection, and the user is sending data.

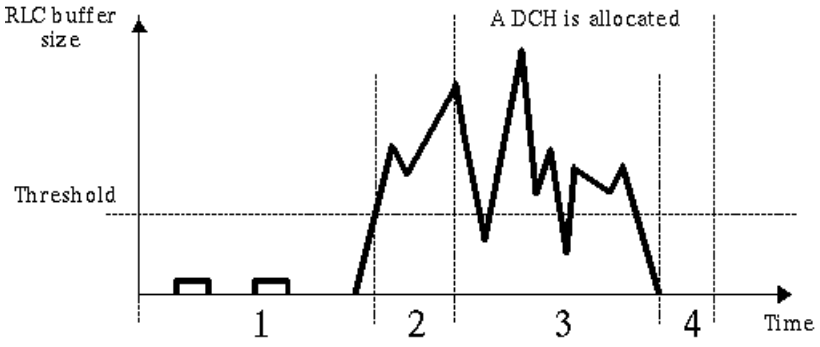

Figure 5.1: DCH allocation with a TCP file transfer.

4. In the last period, the user stopped sending data, but the DCH is still allocated, in case new data would arrive [8]. When there is a very low or no activity on the channel for a certain period, the DCH is released [12]. In the new implementation, the inactivity is detected through a null throughput. This step managed by an "inactivity timer".

There were two different kind of inactivity timers implemented and used during the simulations.

5.2.1 Non-preemptive scheme

The first sets of simulations used a simple non-preemptive scheme for the inactivity timer. It means there is a single inactivity timer at the end of the connection, during which the UE cannot be pulled out of its DCH. This scheme will be called "non-preemptive" [8].

5.2.2 Preemptive/non-preemptive scheme

The last sets of simulations used an adaptation of the preemptive and non-preemptive scheme from [8]. The downswitch step is done through two distinct inactivity timers. The first one is non-preemptive, while the second one is preemptive, meaning that if another UE needs the DCH, it can be released. The first inactivity timer is intended to allow a UE to keep its DCH, even if a TCP RTO occurs (meaning there is still data to send), while the preemptive timer allows a UE to keep its DCH for several consecutive connections, if no other UE needs the bandwidth.

Figure 5.2 shows two UE downloading web traffic and sharing a single DCH. When a DCH is allocated, the intermediate dot shows the end of the non-preemptive inactivity timer and the beginning of the preemptive

inactivity timer. On the first allocation, the green UE is pulled out of its DCH to let the blue connection use it. However, around 150s the green connection tries to use a DCH without success since the blue UE is in its non-preemptive period.

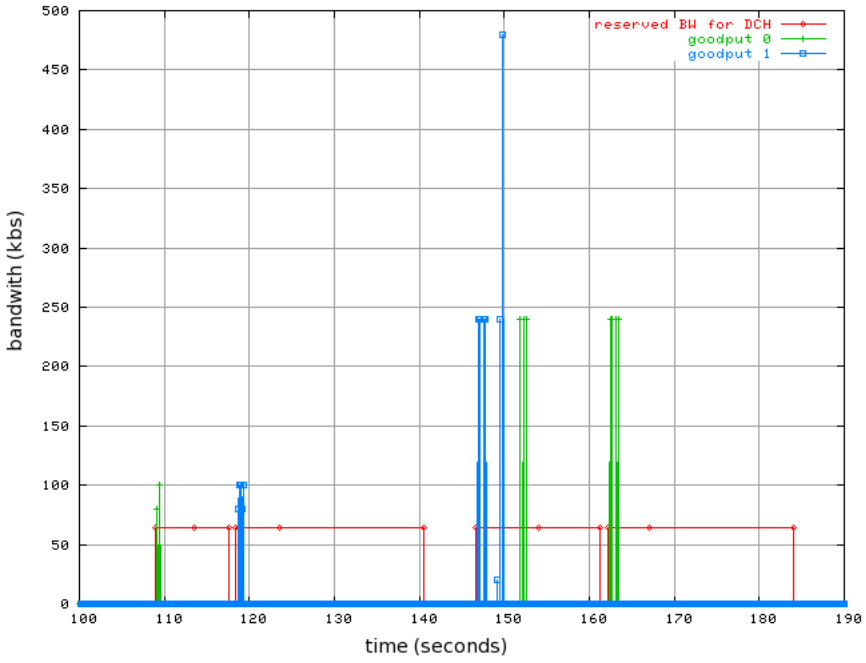

Figure 5.2: DCH allocation (two UE and one DCH)

5.3 WEB traffic generator for NS2

5.3.1 General

Several WEB traffic generators already exist in NS2. However, because of some EURANE particularities, they were not suitable for this project. Therefore, a new web traffic generator for NS2 was implemented, as close as possible to the 3GPP specification [9]. The generated traffic is only downlink, and only acknowledgements are sent from the client to the server.

Figure 5.3 shows the packet trace of a typical web browsing session. The session is divided into packet calls representing a web page download. Each packet call is separated by a reading time, representing the time necessary for the end-user to read the last received web page. A packet call is divided into a main object and several embedded objects (such as pictures).

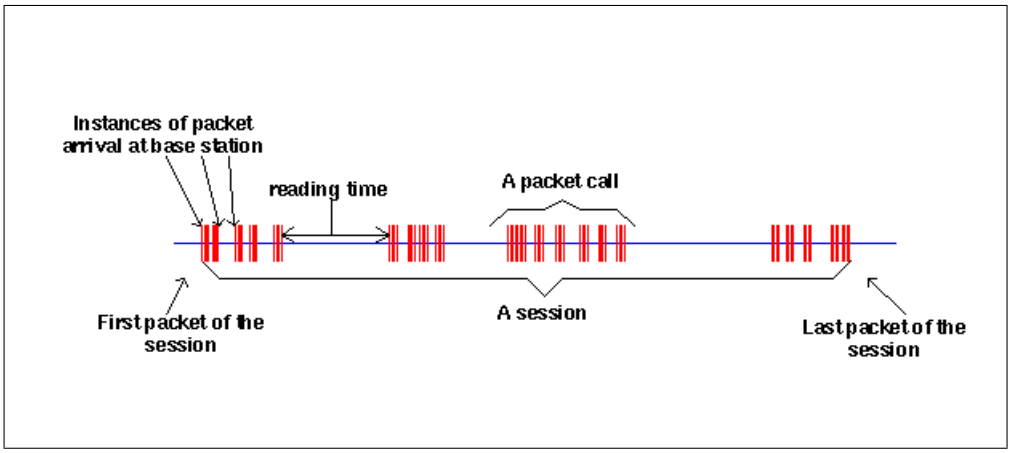

Figure 5.3: Packet Trace of a Typical Web Browsing Session [9].

HTTP1.1 is used to download the objects, meaning that all the objects of a packet call are sent through the same TCP connection. Based on observed packet size distributions, 76% of the HTTP packet calls should use packets of 1,500 bytes, with the remaining 24% of the HTTP packet calls using packets of 576 bytes [9]. These values include the headers.

The distributions of the parameters for the web browsing traffic model are described in Table 5.1 (taken from [9]).

5.3.2 Approach

Each time an object has been sent, a timer is set to decide when the next object will be sent. The way this timer is launched is different if it represents a parsing time or a reading time. If it represents a reading time, the end-user must have received all the data before starting the timer. To ensure this, the traffic generator will wait for all the sent data to be acknowledged, meaning the web page has been successfully received. On the other hand, the parsing time represents the time between packets arrivals at the base station. To stay close to this definition, the traffic generator will wait for an object to be sent (but not necessarily acknowledged) before starting this kind of timer.

Component	Distribution	Parameters
Main object size (Sm)	Truncated Lognormal	Mean = 10,710 bytes Std dev. = 25,032 bytes Minimum = 100 bytes Maximum = 2 Mbytes
Embedded object size (Se)	Truncated Lognormal	Mean = 7,758 bytes Std dev. = 126,168 bytes Minimum = 50 bytes Maximum = 2 Mbytes
Number of embedded objects per page (Nd)	Truncated Pareto	Mean = 5.64, Maximum = 53
Reading time (Dpc) The time between the packet calls	Exponential	Mean = 30 seconds
Parsing time (Tp) The time between the objects in a packet call	Exponential	Mean = 0.13 seconds

Table 5.1: Distributions of the parameters for the web browsing traffic model

Component	Distribution	Parameters
File size (S)	Truncated Lognormal	Mean = 2Mbytes Std dev. = 0.722 Mbytes Maximum = 5 Mbytes
Reading time (Dpc)	Exponential	Mean = 180 seconds

Table 5.2: Distributions of the parameters for the FTP browsing traffic model

5.4 FTP traffic generator for NS2

The main drawback of the existing NS2 FTP traffic generator is that it only sends as much data as it can. However, a FTP session consists of a sequence of file transfers, separated by reading times [9].

The distributions of the parameters for the FTP browsing traffic model are described in Table 5.2 (taken from [9]). The packet size is the same as seen for the WEB traffic.

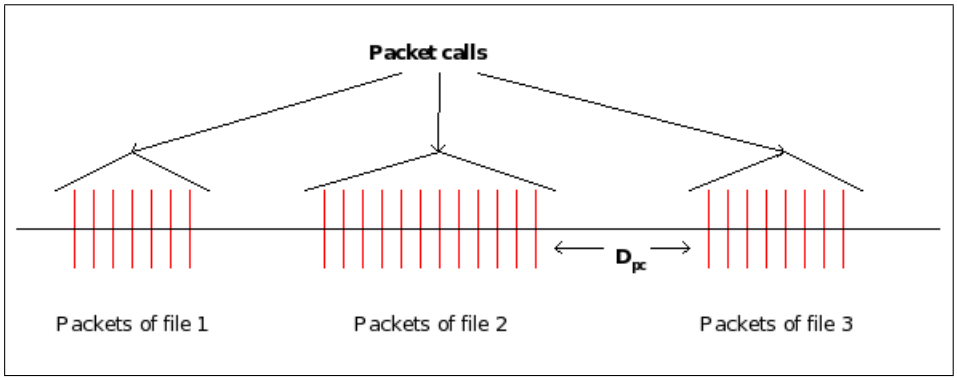

Figure 5.4: Packet Trace of a Typical FTP Browsing Session [9].

5.5 NS2 trace Analyzer

5.5.1 Existing trace analysis tool

Most of the existing tools designed to analyze TCP traffic are first designed to analyze real TCP traffic. Therefore, their approach is to parse and analyze the packet traces from a network or from NS2. This needs sometimes huge trace files, and a difficult way to keep trace of all the packets through hash-tables. Moreover, the NS2 trace files are not in the same format than the classical tcpdump [34], so only a few TCP analysis tools can handle these traces.

The main usable analysis tool is tcptrace [33]. This program was initially designed for real traces, but it is now able to understand NS2 traces. It works well with TCP traces from many simulations, but sometimes gives strange results, and splits a single connection into a lot of different connections. Therefore, it is difficult to trust this kind of results and to use automated scripts.

5.5.2 Parsetcp

During this project, a new TCP analysis tool for NS2 was implemented. It is called Parsetcp. The approach is completely different from the other tools since it does not analyze the packet traces, but directly the NS2 variables. The script was implemented in a shell script with an *awk* parser. The main advantage of this approach is that the variables are only traced when they change, so there are no unnecessary traces and the trace file is small, while the calculated results can be trusted since they directly come from the NS2 implementation. Moreover, the analysis takes a short time, since it is not necessary to have a big hash table or other mechanisms to remember each

traced packet. The results are directly computed by NS2.

Parsetcp is able to give the following results (only about TCP connections):

- *Goodput*: The goodput is the amount of data sent, without headers and retransmissions, per given time.
- *Average outstanding window*
- *Retransmitted packets*
- *Received ACKs*: This value is not used in the case of TCP FACK and SACK, since these TCP versions use a special acknowledgment type.
- *RTT*: Average, minimum and maximum RTT.
- *RTO*: Number of times a RTO occurred.
- *Duplicate ACKs*: The number of times the TCP sender received more than 3 duplicate ACKs in a row.
- *Total packets*: The total number of packets sent (without retransmissions)
- *Transmission time*
- *Connection off time*: When there were more than one connection between the same TCP sender and receiver, this is the time when there was no connection. For example, in the case of HTTP 1.1 traffic, this is the sum of the reading times (see section 5.3).

Note that the goodput, outstanding window, retransmitted packets, average RTT and current RTT evolution can be seen on generated graphs.

All these results are given for each connection, and for the global network utilization. In case there are several connections, the results can be computed using the whole simulation time or only for the time where the connections were used. Finally, it is possible to have an easy to parse output for the results, in order to facilitate the use of automated scripts.

Chapter 6

Simulation results: data transfer using FACH

6.1 Introduction

As seen in the UMTS overview, FACH is a downlink common channel in the UMTS networks. This channel is usually not used to transport data, since a DCH is supposed to be allocated if the receiving buffer becomes bigger than a certain threshold. However, this chapter will describe the hypothetical case of a FTP data transfer over FACH. First, the chosen parameters will be given. Then, the results will be described and explained, and finally a general conclusion will summarize the important results.

6.2 Main parameters

In this simulation, there are two users. The second one begins to transmit over FTP 1,500s after the first one. In order to reference it more easily later in this chapter, "part 1" will mean the first 1,500s and "part 2" will mean the last 6,000s. The maximum number of RLC retransmissions (maxDAT), is taken very high (9,000) in most of the results in order to highlight the congestion problem. The consequence is that no TCP packets are dropped between the RNC and the UE. All the TCP retransmissions occur because of the TCP Retransmission Time Out (RTO), or when the RLC buffer size is full.

Table 6.1 summarizes the used parameters. The error rate does not take care of the correlation between packet losses. The RLC ARQ uses a Selective Acknowledgement mechanism. A RLC POLL timeout of 170ms means that a poll for a bitmap acknowledgment is sent at least every 170ms. The RLC PDU size, POLL timeout and buffer size comes from the EURANE defaults

Main parameters	
Parameter	Value
Number of users	2
Simulation length	7,500s
Traffic	FTP
FACH bandwidth	32kbs
FACH TTI	10ms
RACH bandwidth	16kbs
RACH TTI	20ms
RLC PDU size	40bytes
RLC POLL timeout	170ms
RLC buffer size	100kB
maxDAT	2, 5, 8 or 9,000
BLER	0.01 to 0.9
TCP version	Tahoe, Reno, Newreno, Vegas and Sack

Table 6.1: Summary of the main parameters

while the common channels bandwidth and TTI come from [12]. Note that Fack is not listed in Table 6.1 because of some NS2 implementation problems.

6.3 Results

This section gathers the different simulation results and their explanations. First of all, the evolution of the outstanding window will be shown. Then, different results will be given about the RTT, and the congestion problem at the BS will be depicted. Finally, the impact of maxDAT will be shown. Note that apart from the last subsection, the results will be detailed for a very high maxDAT value (9,000).

6.3.1 Outstanding window

In the particular case where the maxDAT is very large, there are almost no differences between the different TCP protocols. Indeed, if all the retransmissions occur because of the RTO, all the TCP protocols go back to slow start. There is no fast retransmit or fast recovery algorithm used. The only difference we can see concerns the Vegas version, because the RTO is calculated in a different way than the other protocols (see section 3.2.4). Since the only TCP version that has a different behavior is Vegas, the following graphs will only compare TCP Reno and TCP Vegas. Figure 6.1 shows the effect of the congestion on the TCP outstanding window. TCP Reno keeps

the outstanding window size at its maximum value (the advertised window, 20 packets). CWND was never decreased because almost no RTO occurred.

TCP Vegas is aware of the congestion through its enhanced congestion control mechanism (see section 3.2.4). It uses the RTT as a congestion indicator, and not only the lost TCP packets, keeping its outstanding window quite small (2.57 packets). When the second user begins to transmit, Figure 6.1 shows 2 distinct decreasing steps, corresponding to RTOs caused by the congestion, and CWND decreases until half its past value. Knowing that the second user adapts its CWND to the same size, it means that the total outstanding packets on the network remains unchanged.

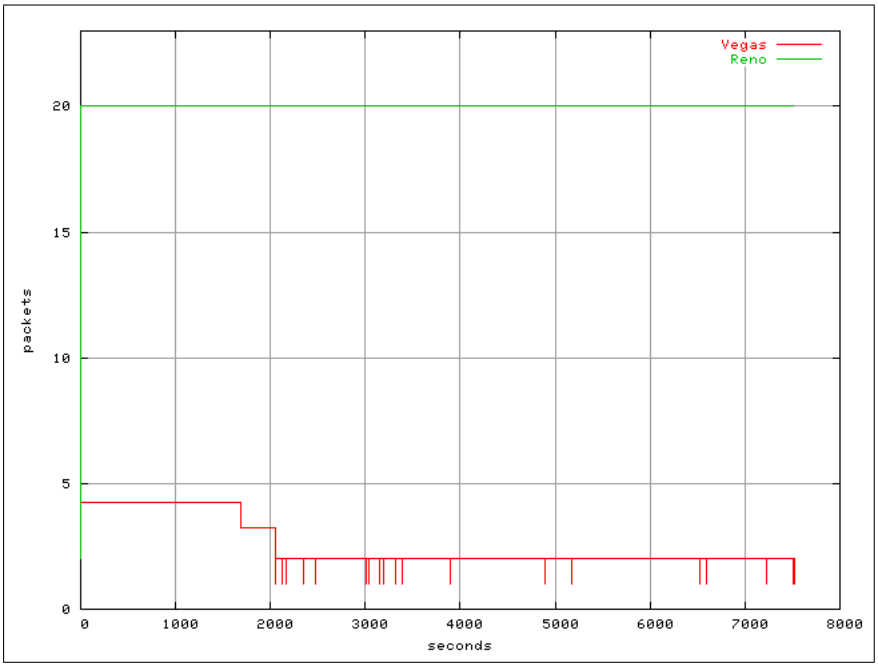

Figure 6.1: First user's TCP outstanding window evolution, 0.01 error rate

6.3.2 Round Trip Time

The following sections will show the relationship between the RTT and some other results. Max DAT is set to 9,000.

RTT, error rate and outstanding window

Figure 6.2 shows the evolution of the average RTT for Reno and Vegas when the first user is alone. While using Reno TCP, the average RTT grows very

fast with the error rate. However, if TCP Vegas is used, the RTT stays quite small and stable until the error rate becomes higher than 0.4.

The RTT is very high when TCP Reno is used. It can be explained by its big outstanding window. Indeed, this window is a good indicator of the packets that are present on the network. If it is too big, the RTT will grow unnecessarily since it will increase the network's congestion, and as seen in the past section, the Reno outstanding window is only limited by the advertised window. The relationship between the TCP advertised window, the RTT and the TCP throughput can be seen as follow [12]:

$$\text{TCP throughput} \leq \frac{\text{advertised window}}{\text{round trip time}}$$

Following this equation, notice that without reducing the throughput, reducing the advertised window lead to a smaller RTT. Figure 6.2 shows that the RTT is always significantly lower in the case of TCP Vegas. This can be explained because of the small Vegas CWND (see section 6.3.1).

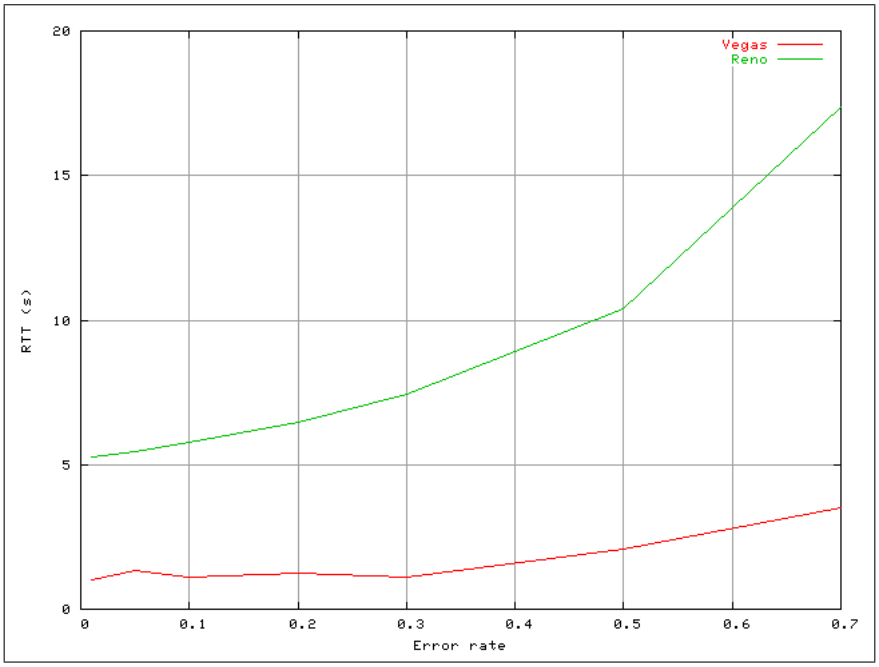

Figure 6.2: Average RTT and error rate for Vegas and Reno TCP, part1

RTT and congestion

The congestion at the BS has a big impact on the RTT. Figure 6.3 shows the SRTT (see section 3.1.3) evolution with a small error rate (0.01). It is

obvious that the negative impact of the congestion on the SRTT is bigger while using TCP Reno than while using TCP Vegas. Moreover, comparing Figure 6.3 and Figure 6.1 between 1,000 and 2,000 seconds confirms that the CWND and the RTT are closely related. The two decreasing steps of the Vegas CWND results in two distinct decreasing steps of the Vegas SRTT. In the case of Reno, the RTT grows very rapidly and never goes down, since the congestion is not detected by the Reno TCP sender. Therefore, the CWND does not adapt itself and the number of outstanding packets is doubled, resulting in a lot of local retransmissions and in a higher RTT.

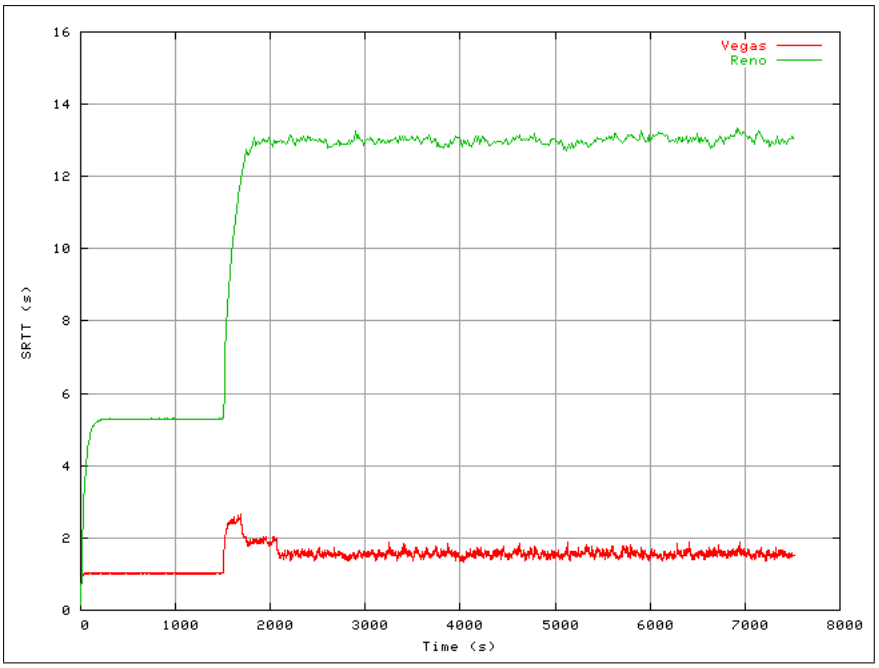

Figure 6.3: SRTT and congestion at the BS, 0.01 error rate

6.3.3 Retransmitted packets

As said before, in the case of Reno, Newreno, Fack, Sack and Tahoe, there are almost no retransmissions, even if the error rate is high. This happens because there is a huge maxDAT (9,000). However, there are more retransmissions in the case of Vegas because the RTT tends to be very high. This results in unnecessary retransmissions, but when the error rate is not big, the RTT is a good way for the TCP sender to be aware of the congestion and to reduce its CWND.

6.3.4 Congestion and error rate

This section intends to show the congestion problem at the BS. First, it will be depicted with a small error rate. Afterwards, the impact of the error rate on this congestion problem will be explained. Max DAT is set to 9,000.

Small error rate (0.01)

Figure 6.4 shows the total TCP goodput and the RLC throughput with a 0.01 error rate. The RLC throughput is the throughput measured on the downlink, between the RNC and the BS. This throughput includes the TCP and RLC headers, the TCP and RLC retransmissions and the RLC status messages.

During the first part, the link is almost used at its full capacity (32kbs). The TCP Vegas goodput is a little bit higher than the other protocols (31.48kbs instead of 30.36kbs), and the RLC throughput is the same. This is explained by the small Vegas window (4 TCP segments). Indeed, with a too high window (19.97 TCP segments for TCP Reno), the UE alone can overflow the BS RLC buffer. This causes some RLC packets to be retransmitted, a longer RTT and therefore a lower goodput.

During the second part of the simulation, the difference between the RLC throughput and the goodput becomes really big, and the goodput decreases. Moreover, the RLC throughput is a lot higher than the link capacity. This is explained because the RLC throughput is monitored on the Iub (between the RNC and the BS), which is a very fast link, and that all the congestion losses occurs at the BS. Therefore, the RLC throughput contains all the RLC retransmissions. As there is almost no limit on the number of such retransmissions, the buffer at the BS is continuously overwhelmed. This explains the smaller TCP goodput (25.76kbs for Reno, 26.72kbs for Vegas). Another observation is that the RLC throughput while using Vegas is smaller than while using Reno. As explained before, the Vegas new congestion control is based on the RTT, thus Vegas is aware of the congestion at the BS. Vegas considers that some TCP packets are lost, retransmits them and decreases its CWND. This results in some unnecessary TCP retransmissions and in a smaller amount of RLC retransmissions.

High error rate

When the error rate becomes higher, the general behavior does not change. Figure 6.5 shows the evolution of the goodput following the error rate. The general trend is that the goodput decreases while the error rate grows. It is logical since each packet may have to be retransmitted several times by the RLC. Moreover, the goodput while using Vegas is always higher than

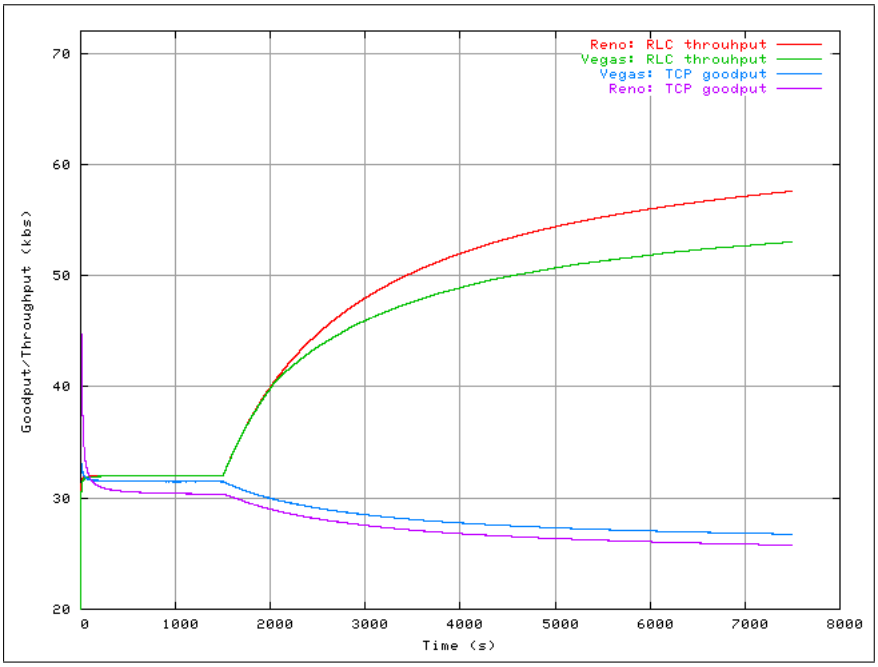

Figure 6.4: Congestion at the BS, 0.01 error rate

the goodput while using Reno, for error rates smaller than 0.5. Afterwards, the Vegas performance becomes worse. The reason is that for an error rate higher than 0.4 or 0.5, the RTT with Vegas TCP becomes higher (see section 6.3.2), leading to more unnecessary TCP retransmissions.

As seen in the previous section, the congestion problem at the BS results in a decreasing total goodput and an increasing RLC throughput. However, the goodput difference between part 1 and part 2 (respectively with and without congestion) is decreasing with the error rate. Figure 6.6 shows the evolution of this goodput difference following the error rate in the case of TCP Reno. As this difference is decreasing while the error rate grows, the congestion is a more important problem when the error rate is small. This difference is decreasing because there are more local retransmissions, leading to a bigger RTT. A high RTT causes more RTO to occur, forcing the TCP sender to reduce its sending rate and to go back into Slow Start.

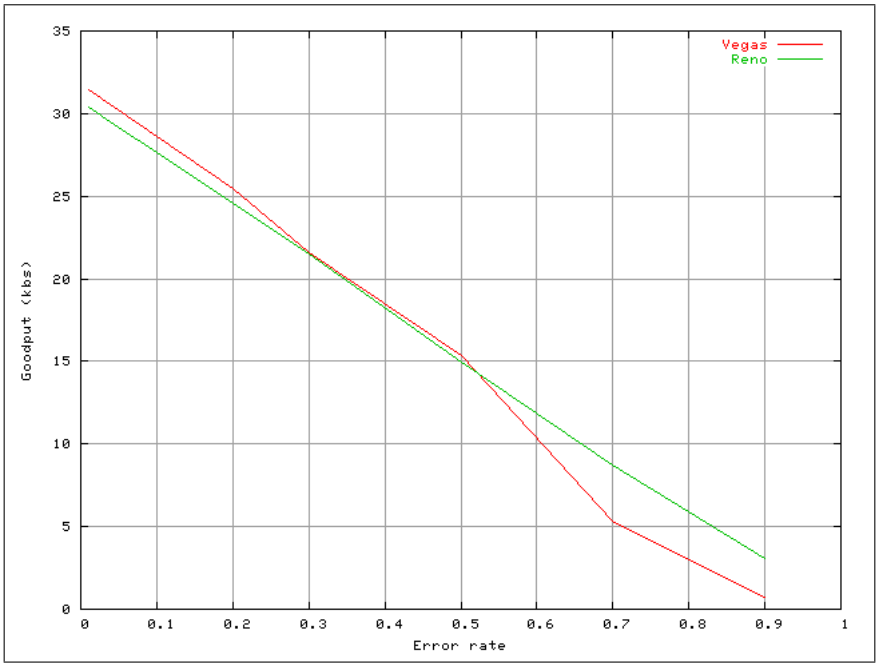

Figure 6.5: TCP goodput and error rate for TCP Reno and Vegas, part 1

6.3.5 Impact of maxDAT

This section shows the impact of a maxDAT limitation on the goodput, when the user is alone (first part of the simulation). The congestion problem will not be explained since a large inequity occurred between the connections when the second user begins to transmit. This inequity is caused by the DropTail queue at the BS. This unfairness of TCP under DropTail FIFO queueing was already mentioned in some studies [32]. It has been observed here that the inequity problem becomes bigger when the link bandwidth decreases. Active queue management could be implemented to reduce this problem [31].

Figure 6.7 shows the impact of the error rate on the goodput, if maxDAT is small (2). In this case, the goodput falls rapidly with the error rate, and becomes null when the error rate is bigger than 0.2. This general trend is logical since if a lot of TCP packets are dropped, TCP will understand it as a congestion indication and slow down its sending rate. When the error rate becomes really high, a lot of RTO will occur and finally almost no packets will be sent.

If the error rate is very small (0.01), there are almost no TCP packets

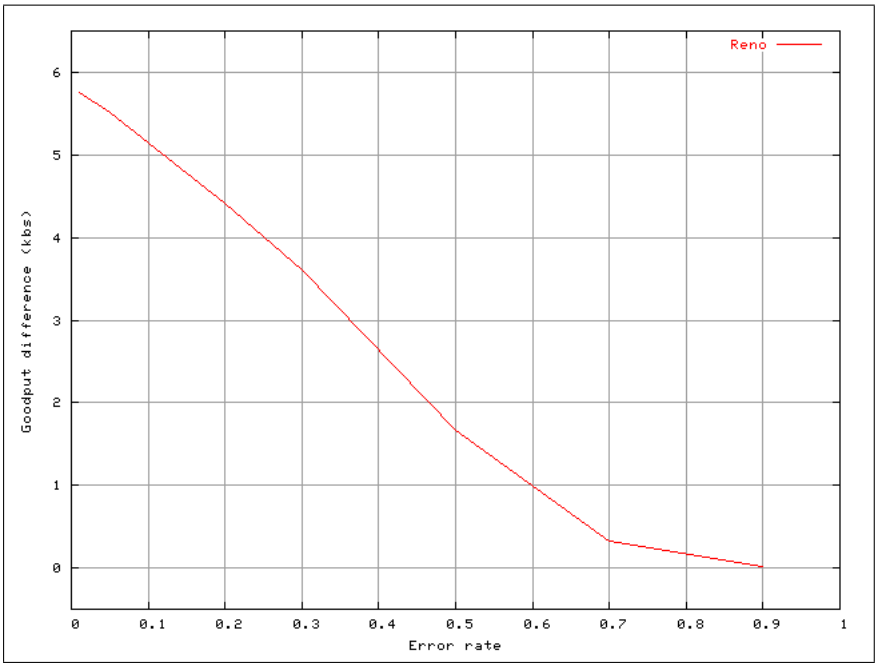

Figure 6.6: Global goodput difference between part 1 and part 2 following the error rate (TCP Reno)

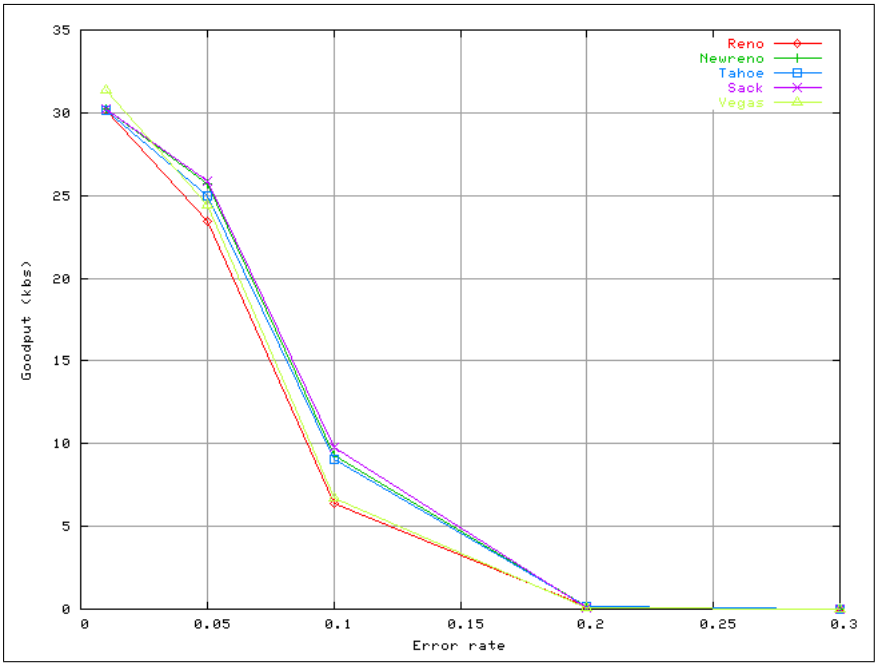

Figure 6.7: Goodput and error rate, maxDAT = 2, part 1.

dropped. Therefore, Reno, Newreno and Tahoe perform the same, as no congestion control is involved. Sack performs the same as these protocols. Indeed, there are almost no lost packets so the SACK mechanism does not avoid a lot of unnecessary retransmissions. Moreover, other studies showed that in general, using Sack does not represent a significant improvement in lossy links [31].

TCP Vegas performance is a little bit higher, since it reduces its CWND following the RTT and keeps it smaller than the other TCP versions (see Table 6.2). A smaller RTT usually leads to a higher throughput (following the equation given in section 6.3.2).

When the error rate grows, there are more losses so the TCP versions behavior changes. TCP Vegas does not achieve the best performance anymore, since the other versions also reduces considerably their CWND (see Table 6.2), achieving a smaller RTT. Vegas performance is then similar to Reno performance. They both use a standard Fast Recovery mechanism that lead to a stall when there are a lot of packet losses in the same window (see section 3.2 for more details).

Sack, Newreno and Tahoe give a similar performance, but Sack achieved the best performance thanks to its Selective Acknowledgement mechanism. Newreno follows and is just a little bit better than Tahoe thanks to its Fast Recovery functionality. When the error rate is higher than 0.2, all the protocols achieved an almost null goodput.

Figure 6.8 shows the impact of maxDAT on the goodput, if the error rate is equal to 0.05. The difference between the TCP versions when maxDAT is small has already been explained in this section. However, this figure shows that above a certain value of maxDAT, Vegas is the only TCP version that achieves a different performance. TCP Vegas performance is better than the others since it keeps its CWND small, as seen previously in this chapter.

6.4 Conclusion

This chapter showed the impact of the congestion at the BS, and the fact that an infinite RLC retransmission limit hides the congestion to the TCP sender. This problem leads to a smaller goodput, a lot of RLC retransmissions and a very high RTT. However, when there is no congestion, it was shown that a higher maxDAT leads to a higher goodput, whatever TCP version is used. The simulations also show that an infinite maxDAT value is not needed in order to reach the maximum goodput when the user is alone. The congestion problem with a small maxDAT value was not showed here

BLER	TCP version	Average window (packets)	Average RTT (s)
0.01	Tahoe	16.54	4.17
0.01	Reno	18.47	4.71
0.01	Newreno	18.44	4.69
0.01	Vegas	3.31	0.812
0.01	Sack	17.93	4.54
0.01	Fack	18.13	4.52
0.1	Tahoe	2.15	0.70
0.1	Reno	2.12	0.75
0.1	Newreno	2.16	0.69
0.1	Vegas	2.54	0.67
0.1	Sack	2.18	0.68
0.1	Fack	2.13	0.69

Table 6.2: Average outstanding window and RTT for maxDAT = 2 (part 1)

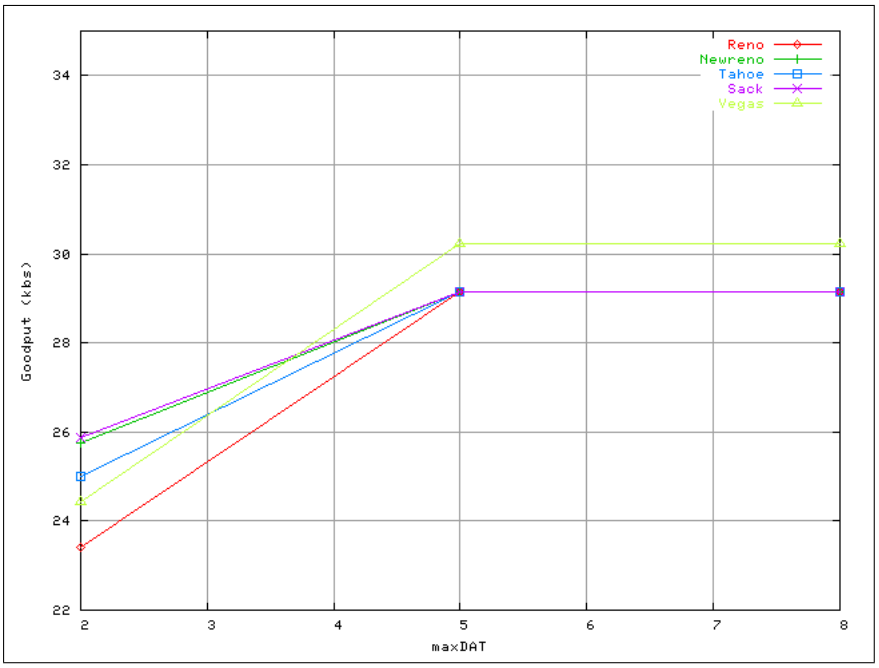

Figure 6.8: MaxDAT and Goodput, 0.05 error rate, part 1.

because of the DropTail FIFO queueing unfairness at the BS.

In case there are some TCP packets dropped due to the BLER (thus with a small maxDAT) and no congestion, Sack and Newreno achieve the higher goodput. Tahoe just follows while Vegas and Reno give a smaller goodput because of the stall that is a well known problem of their Fast Recovery mechanism.

When maxDAT is very large, the best performance is achieved by Vegas that keeps a small outstanding window, and that adapts its sending rate accordingly to the current congestion. The other TCP versions give similar performance.

Following these observations, to set maxDAT to a high value and to use TCP Vegas seems to be the best combination to ensure the equity between the connections and the highest possible goodput.

Chapter 7

Simulation results: Web traffic using the FACH/DCH switch, with a non preemptive inactivity timer

7.1 Introduction

Most probably, a major fraction of the UMTS data traffic will be WEB traffic, which is characterized by some bursty periods (see section 5.3 for more details). The connection setup procedure and some small data can be transmitted using the common channels (FACH/RACH) while a DCH is required to transmit large volumes of data [12] (see section 5.2). The goal of this chapter is to analyze the impact of the main RLC parameters on the performance, using different TCP versions. First, the parameters to be considered and the performance metrics to be used will be introduced. Then, the results from the simulation runs will be discussed.

7.2 Simulated scenarios, parameters and performance metrics

The scenarios consider 30 users downloading WEB data in one cell. The users begin using the common channels, and when the RLC buffer grows above the upswitch threshold, a switch to a DCH is triggered. Each simulation is ran 3 times for 4,000 seconds, with each time a different seed for the random variables used in the traffic and error generators. Then, the results were averaged to get statistically sounded values.

Gilbert model statistics, taken from [25]			
State i	Pr(i)	$Pr(1\|i)$	$Pr(0\|i)$
0	0.9449	0.0087	0.9913
1	0.0551	0.8509	0.1491

Table 7.1: Multistate error model parameters

The error rate is assumed to be constant, but a two-state Gilbert model from [25] is used to correlate packet losses. In the used Gilbert model, there is an error-free state (0) and an error state (1). Transitions between the states are governed by probabilities. Table 7.1 shows their used values, where $Pr(A|B)$ is the probability to go to the state A if the current state is B.

The corresponding error rate is calculated as follow:

$$Pr(error) = \frac{Pr(1|0)}{Pr(1|0)+Pr(0|1)} = 0.055$$

The parameters of interest are maxDAT (taken between 2 and 6), the upswitch buffer threshold (500 bytes, 1,000 bytes or 1,500 bytes) and the single non-preemptive inactivity timer for the downswitch (2, 4, 6, 8 or 10 seconds). Figure 7.1 summarizes the non-preemptive scheme described in Section 5.2. No preemptive inactivity timer was used for this chapter simulations. The TCP versions that are considered in this chapter are Tahoe, Reno, Newreno and SACK. TCP FACK and Vegas are not listed, as the simulation results for these two cases did not succesfully complete due to unexpected problems in the VEGAS and FACK NS2 implementation. The encountered TCP Vegas implementation problem is specific to the WEB traffic generator. Table 7.2 summarizes the values of the parameters for the simulations. The parameters used in the WEB traffic generation are listed in Section 5.3.

The allocation time, the inactivity timer values, the common channels parameters, the DCH downlink bandwidth and the maximum cell downlink bandwidth come from [12]. The DCH uplink bandwidth is normally 16kbs for a 64kbs downlink bandwidth [12]. However, its value was set to 32kbs in order to make sure that the only bottleneck in the connection would be the downlink, since this study is limited to the downlink performance. The RLC ARQ uses a Selective Acknowledgement mechanism. A RLC POLL timeout of 170ms means that a poll for a bitmap acknowledgment is sent at least every 170ms. The other parameters are set to the NS2 or EURANE default values.

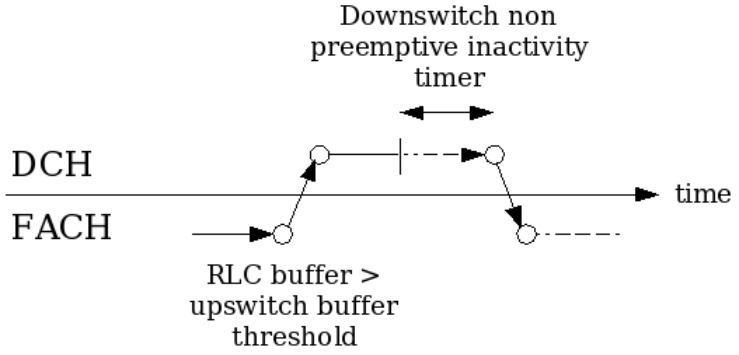

Figure 7.1: DCH allocation with a single non-preemptive inactivity timer.

Main parameters	
Parameter	Value
Number of users	30
Traffic	WEB
Allocation time	0.9s
Error rate	multistate
Pr(Error)	0.055
maxDAT	2,3,4,5 or 6
Upswitch buffer threshold	500, 1,000 or 1,500 Bytes
Inactivity timer	2,4,6,8 or 10s
TCP version	Tahoe, Reno, Newreno, Sack
Simulation length	4,000s
Number of simulations	3
FACH bandwidth	32kbps
FACH TTI	10ms
RACH bandwidth	16kbps
RACH TTI	20ms
DCH downlink bandwidth	64kbps
DCH downlink TTI	10ms
DCH uplink bandwidth	32kbps
DCH downlink TTI	10ms
RLC PDU size	40 Bytes
RLC POLL timeout	170ms
RLC buffer size	100kB
Max cell downlink bandwidth	800kbps

Table 7.2: Summary of the main parameters

The simulation results were analyzed using the following performance metrics:

1. Average RTT: the average Round Trip Time in seconds.
2. Retransmitted packets: the total number of retransmitted packets.
3. Cell efficiency and occupancy: the cell occupancy is the reserved bandwidth of the cell (including the FACH and all the allocated DCHs). This value is not a good performance indicator, as it shows the reserved bandwidth, but not the used bandwidth. Instead, we introduce a new metric called Cell Efficiency, which is defined as:

$$\text{Cell efficiency} = \frac{\text{Average goodput}}{\text{Average cell occupancy}}$$

4. Goodput: the total goodput transmitted in the UMTS network, in kbps. This goodput does not include the TCP and RLC retransmissions, nor any header.
5. Full-cell time: the time (in seconds) where the cell was fully loaded and thus where no new DCH could be allocated.

7.3 Results

The following sections will explain the most important results, and the impact of the different parameters on these results. When the TCP version is not specified on a graph, the curve is an average of all the TCP versions, in order to show the general trend.

7.3.1 Average Round-Trip Time

The average RTT is always between 400ms and 700ms. These values are large compared to the ones given in [12] (more or less 200ms for a 128 Bytes ping on DCH, 285ms for a 32 Bytes ping on FACH), but it can easily be explained. First, the average packet size (1,278 Bytes) is a lot larger than in these examples. Second, there are a lot of connections with only a few packets, and a high probability that a DCH is allocated for each connection. Each time a DCH is allocated, 900ms are added to the RTT of the packet(s) in the buffer. Therefore, the RTT of these first packets is most probably higher than one second, while the other packets have a smaller RTT if the congestion is low.

Another interesting thing about the RTT is that the RTT is not a good performance indicator as goodput is not monotonic increasing function of RTT. Nevertheless, a large RTT is often associated with a high goodput (see Figure 7.2). When the RTT is large, that means that there were probably a lot of DCH allocation (it takes 900ms). Moreover, DCH allocation could lead to congestion at the BS if all DCHs are allocated. Therefore, in the case of web traffic, a large RTT is often related with a high goodput.

Figure 7.2 can be divided in three parts. First, the goodput is quite high for a small RTT. It means that there were only a few RLC retransmissions (see Figure 7.3) and a few DCH allocations. The goodput is high because the connections tended to stay in DCH. The second part shows a dip (less than 10%) in the goodput when the RTT becomes a little bit higher. A more precise analysis of the numeric results shows that it happens when the upswitch threshold is high, thus when the connections tend to be blocked in FACH. The third part, where the RTT and the goodput are the highest, contains all the other simulations. This general behavior is the same for all the TCP versions.

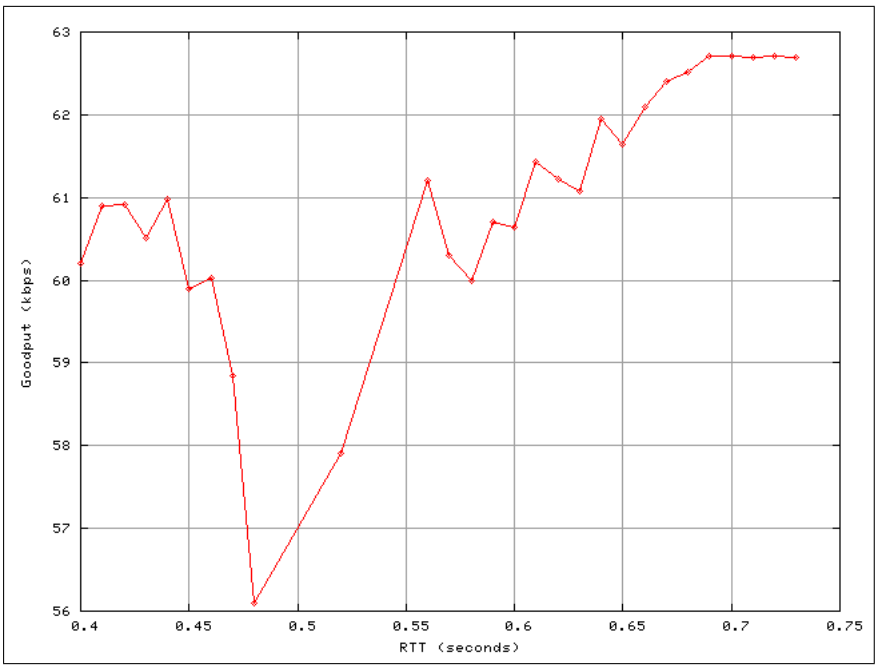

Figure 7.2: Goodput and RTT

Figure 7.3 shows that maxDAT has a big impact on the RTT. It is logical since each retransmission consumes some time. Moreover, the effect

TCP version	Average Number of TCP retransmissions per connection	Average Number of RTO per connection
Tahoe	1.76	0.9
Reno	1.77	0.9
Newreno	1.76	0.9
Sack	1.74	0.9

Table 7.3: Retransmission statistics

of increasing maxDAT is more pronounced at lower maxDAT values than at higher ones. This is due to the fact that the BLER is small, so a small maxDAT is sufficient to hide the unreliability of the wireless link to the TCP sender.

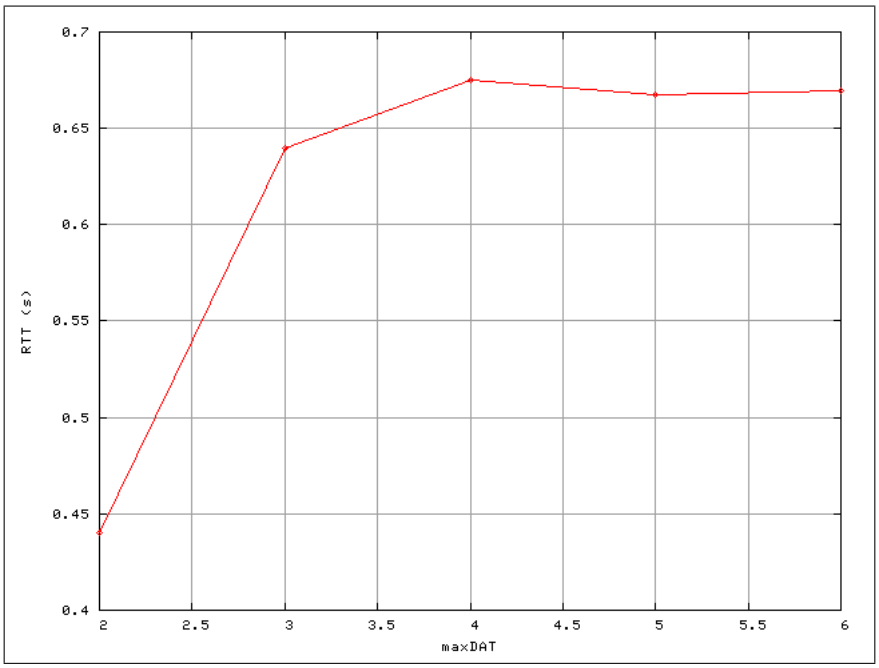

Figure 7.3: maxDAT and RTT

7.3.2 Retransmitted packets

Retransmissions statistics averaged over all simulations are given in Table 7.3. There are less than two retransmissions per connection, and at least half of them are caused by a RTO. This can be explained by the fact that a DCH is almost always allocated to a connection, and not necessarily at the beginning if the upswitch threshold is higher than the packet size. The

DCH allocation adds 900 ms to the RTT of the packet(s) belonging to this connection in the buffer. Therefore, the RTT of these packets becomes very large and a RTO can occur even if the packets are not really lost. Moreover, Figure 7.4 shows that the number of TCP retransmissions per connection decreases very quickly when maxDAT grows. All the TCP versions react in the same way, but SACK has always a lower number of retransmissions since it uses a selective repeat mechanism.

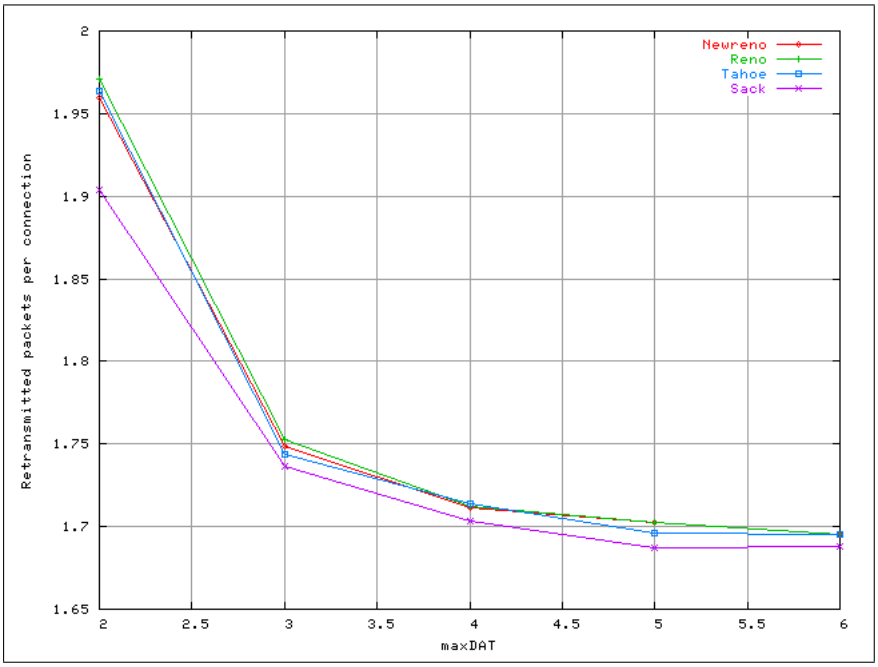

Figure 7.4: maxDAT and retransmissions per connection

Figure 7.5 shows the number of RTO per retransmitted packets, compared to maxDAT. The number of TCP retransmissions per connection decreases very quickly when maxDAT grows. Comparing TCP versions, the number of RTOs when Tahoe is used is high when maxDAT is small, because the errors are due to the BLER and Tahoe is falling back into Slow Start every time a loss occurs. Thus, the CWND stays too small and there is only a small probability that more than 3 packets arrive at the receiver after a loss occurs (generating 3 duplicate ACKs). Most of the losses have to be detected through a RTO. The general behavior of Reno and Newreno is more or less the same as Tahoe. The difference is that, thanks to the Fast Recovery mechanism, their CWND is not reduced each time a packet loss is detected. Therefore, more packets are sent at the same time and the probability of having 3 duplicate ACKs instead of a RTO is higher for a

small maxDAT.

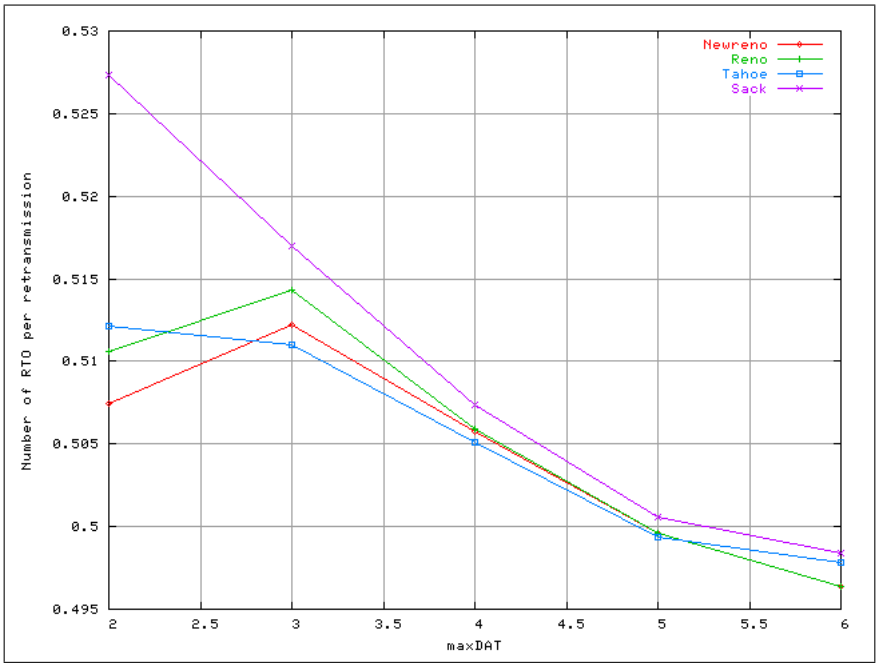

Figure 7.5: maxDAT and RTO

There are less RTOs when maxDAT is high. For a RTO to occur, a sequence of sent packets must have been lost. Otherwise the loss of a single packet would have been detected through 3 duplicate ACKs. Therefore, if maxDAT is high, the probability that a sequence of TCP packets is lost becomes very low.

The proportion of packets retransmitted because of a RTO is always a lot higher in the case of SACK, especially when maxDAT is very small. This can be explained by the fact that SACK only detects the loss of a retransmitted packet through a RTO mechanism, not 3 duplicate ACKs [22]. Therefore, if there are many losses (partially due to the BLER), a lot of retransmitted segments could be lost and SACK would have to wait for their RTO.

When maxDAT grows, the congestion problem at the Node B is a little bit hidden by the RLC retransmissions. Therefore, if a loss is detected, even through 3 duplicate ACKs, it means that the congestion at the Node B is very high. As Reno and Newreno do not reduce abruptly their sending rate by going into Slow Start (as Tahoe), they tend to send too many packets and it leads to an even bigger congestion, generating more RTOs. Moreover, a

greater number of RTOs occur in Reno than Newreno, especially if maxDAT is small, because of the stall problem which is a characteristic of Reno (see Section 3.2 for more details).

A common way of optimizing the goodput in wireless networks is to initialize the TCP sender window at a high value [12]. Although it was not simulated for this report, we believe that this kind of optimisation is probably not a good idea when the FACH/DCH switch occurs. Indeed, if the first packets sent are retransmitted because of the DCH allocation time, it will lead to unnecessary retransmissions, and the TCP window will be decreased anyway before new packets could be sent.

7.3.3 Cell efficiency

In general, for identical performances, it is better to chose a parameter combination that gives a high cell efficiency. If the average cell occupancy is small, the cell is most probably able to accept more users than if the average cell occupancy is high. Practically, the simulations show that the efficiency stays between 9% and 20%. These small values are explained by the downswitch inactivity timer, where the DCHs are still reserved while the connection is inactive. The connections are usually short, and the packets are not necessarily sent continuously because of the parsing time. It is impossible to obtain a 100% efficiency because of the retransmissions, of the inactivity timer and of the headers.

The cell efficiency is strongly dependent on the inactivity timer (see Figure 7.6). Therefore, it is better not to use a too long inactivity timer. However, this conclusion may not necessarily be correct with a non-preemptive inactivity timer. Indeed, it might be useful to downswitch an inactive connection to FACH before the end of the inactivity timer if an active connection is waiting for an upswitch. This issue will be investigated in the following chapter.

7.3.4 Goodput

In this section, we will see the general impact of maxDAT, the upswitch threshold and the inactivity timer on the goodput.

maxDAT

Figure 7.7 shows the effect of maxDAT on the goodput. It can be seen that it is better to have a high maxDAT. If some packets are dropped because of an error, TCP would think it is because of congestion, retransmit them and then decrease its congestion window (see Chapter 3). Moreover, a local

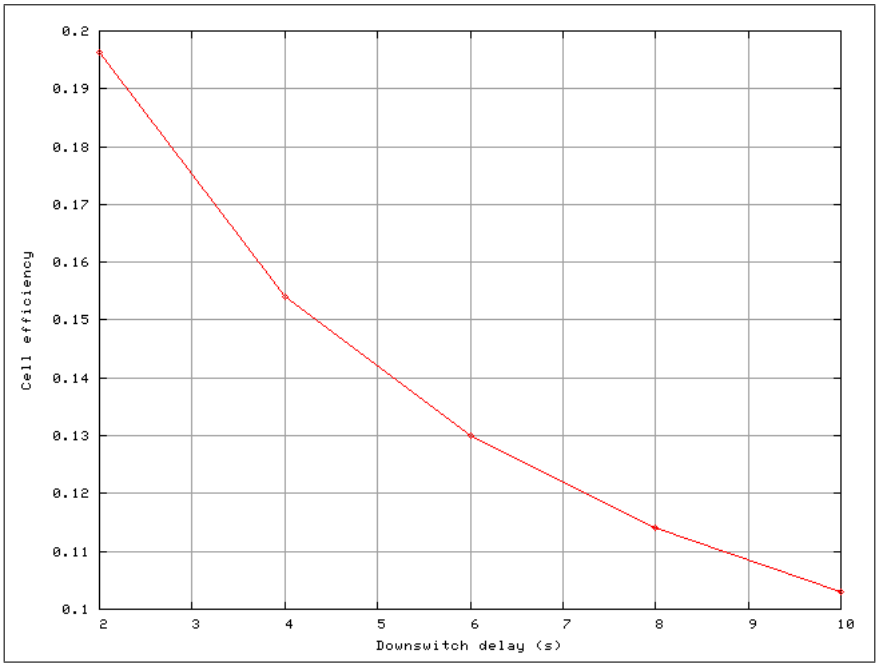

Figure 7.6: Inactivity timer and cell efficiency

retransmission of a small RLC PDU is faster than an end-to-end retransmission of a big TCP segment. The congestion problem shown in Chapter 6, is not really important in this case, since only a few packets should be sent over FACH if a DCH is allocated.

When maxDAT is 2, all the versions achieve more or less the same performance. As seen in section 7.3.2, there are a lot of retransmissions, meaning that the CWND does not have the time to grow. If CWND is small, the Fast Recovery mechanism of Reno and Newreno does not give better performance than a Slow Start. Reno and Newreno performances are even a little bit worse than Tahoe since they go out Fast Retransmit faster than Tahoe from Slow Start. The gain in performance of SACK thanks to its Selective Repeat is low with WEB traffic, since there are only a few packets per connection (the average is 8). If there are only a few packets to send, the TCP sending window is never high, and there is only a few unnecessary retransmissions with the other versions[30]. Moreover, this small advantage is canceled by the high proportion of RTO of Sack, as seen in section 7.3.2.

If maxDAT is equal to 3, we can see significant differences. Reno and Newreno have the same behavior, and give better performance than Tahoe. These three TCP versions do not have a Selective Repeat mechanism and

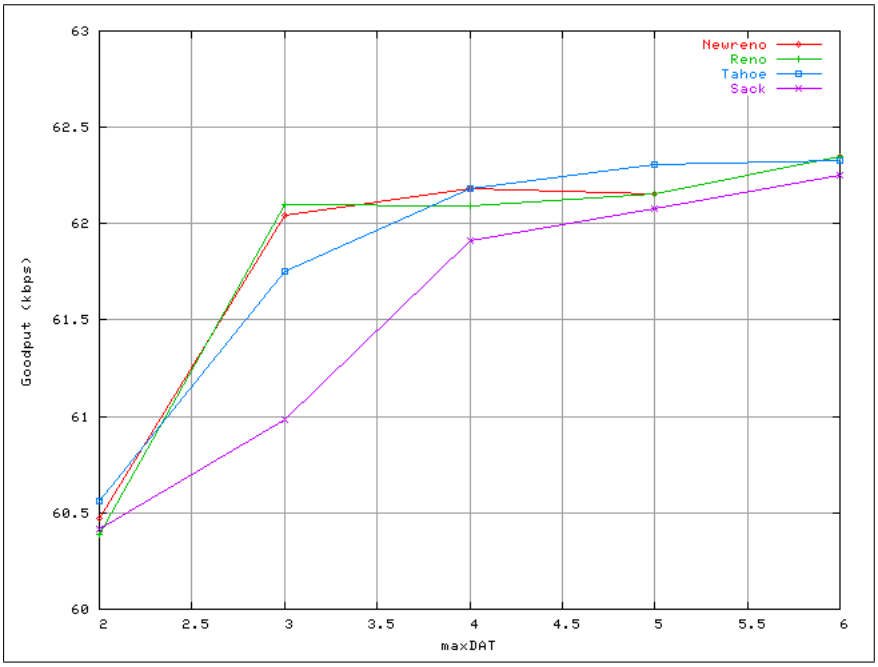

Figure 7.7: Impact of maxDAT on the goodput

may retransmit more than one packet for each loss. This can lead to unnecessary retransmissions, but in case the upswitch threshold is bigger than the TCP packet size, the RLC buffer is filled and a DCH allocation may be triggered. As the Fast Recovery mechanism of Reno and Newreno is more aggressive than Tahoe and its Slow Start, the buffer is filled faster, leading faster to a DCH allocation and thus to a higher goodput. In the SACK case, as it only retransmits the lost packet, the RLC buffer does not grow and no DCH is allocated because of the retransmissions.

Finally, if maxDAT is equal to or higher than 4, some TCP losses can still occur. However, if even a single TCP packet is lost, it means the congestion at the BS is really big, since the number of local retransmissions is high. Therefore, the arrival of 3 duplicate ACKs is an indication of heavy congestion and not of small congestion as with a small maxDAT. It is better then to reduce abruptly the sending rate to reduce the congestion problem. In this case, Tahoe gives better performance than Newreno and Reno, since it always reacts to the congestion by going back into Slow Start. When maxDAT becomes higher, all the TCP versions tend to achieve the same performance since there are only a few dropped packets. SACK stays below the other TCP versions for the same reasons as mentioned before.

Upswitch threshold

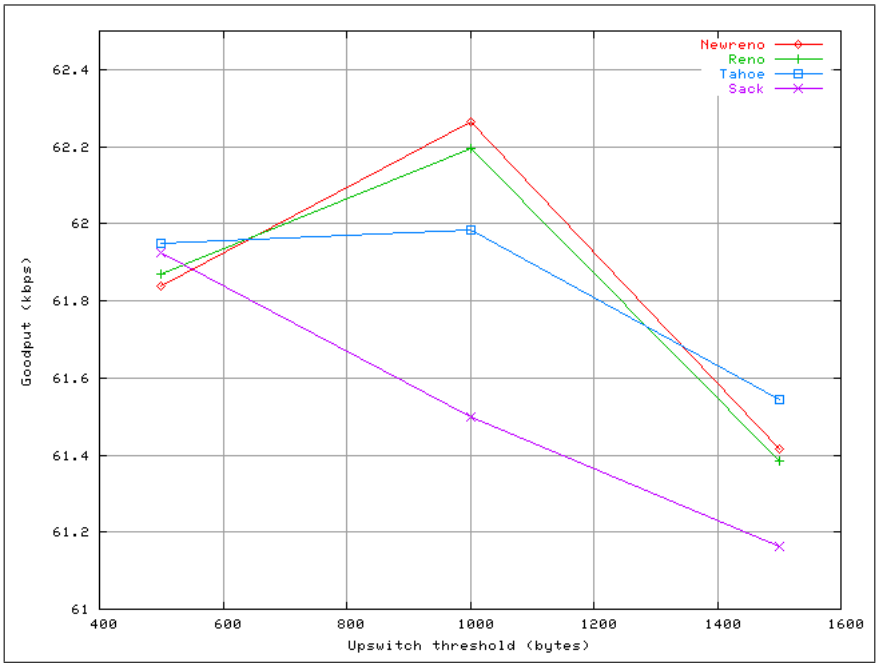

Figure 7.8: Impact of the upswitch threshold on the goodput

Figure 7.8 shows TCP goodput as a function of the upswitch treshold. The larger threshold gives the worst performance, but the optimal threshold is not necessarily the smallest, and this depends on the TCP version. If a connection has only a few small packets to transmit, the DCH allocation time can make it worse to use a dedicated channel than to send everything using the common channels. Moreover, if the downswitch delay is high and the upswitch threshold is small, the DCH will be allocated too easily for a long time to connections that do not necessarily need it, while some others will have to stay on FACH because all the DCHs are allocated. Figure 7.9 shows that a small threshold leads to a bigger full-cell time, where the full-cell time is the time (in seconds) where the cell was fully loaded and thus where no new DCH could be allocated.

When the upswitch threshold is smaller than the smallest TCP segment, even the very small connections try to use a DCH, hence saturating the cell. A full cell prevents the other connections from triggering a switch to a DCH. Some large connections may thus have to send data over FACH. In this case, SACK achieves the best performance, thanks to its Selective Repeat mechanism. As the congestion at the Node B is partially hidden by the local retransmissions, to receive 3 duplicate TCP ACKs means that the

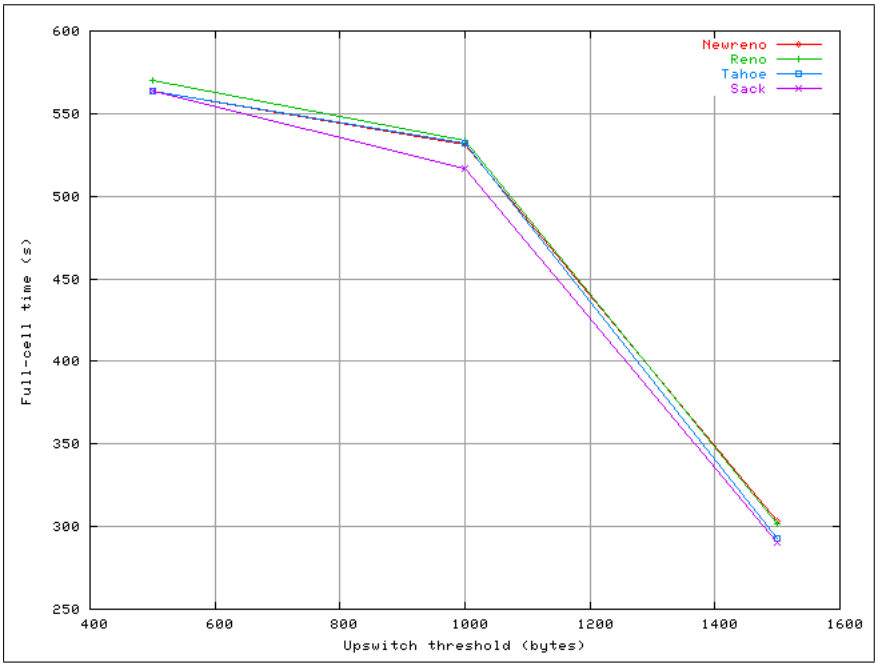

Figure 7.9: Impact of the upswitch threshold on the full-cell time

congestion is very severe. Therefore, it is better to reduce the flow abruptly like Tahoe, in order not to overwhelm the network, than to use a Fast Recovery mechanism. As a result, Newreno achieves a worse performance than Reno, since its Fast Recovery is more aggressive.

When the upswitch threshold grows, the Newreno and Reno performance increase very fast until 1,000 Bytes, and then Newreno becomes better than Reno. If the upswitch threshold is larger than some TCP segments, a high number of fast retransmissions can quickly fill the RLC buffer and then trigger a switch to a DCH. Therefore, the more aggressive the Fast Recovery is, the best performance the TCP version achieves. The Tahoe performance stays more or less stable since it retransmits the same number of segments but in a slower way, due to the fact that it always goes back into Slow Start (see Section 3.2). Therefore, a switch to a DCH is triggered, but not as fast as for Reno and Newreno. Finally, SACK performance decreases very fast since it almost never trigger a switch because of its small number of retransmissions.

When the threshold reaches the size of the largest TCP segments, the connections tend to use FACH for a long time before a DCH could be allocated (if ever allocated). Tahoe still outperforms Reno and Newreno, and

Newreno remains better than Reno because many DCHs are allocated due to its more aggressive retransmission mechanism. SACK still achieves the worst performance.

Inactivity timer

Finallly, Figure 7.10 shows the effect of the inactivity timer on the goodput. This figure was averaged for all the protocols in order to show the general trend. Otherwise, the values fluctuated a lot. These unpredictable fluctuations are due to the packet scheduler that is very simplistic. In general, a high inactivity timer value helps a connection to remain in DCH and therefore increases its goodput. However, it is clear that a too long inactivity timer has a bad impact. Indeed, there is a higher probability for the cell to be fully loaded if the inactivity timer is long (see Figure 7.11), even if the DCH is not effectively used. Thus, the connections that really need a DCH may not have one. It could also cause unfairness between the connections, but this problem was not tackled in this project.

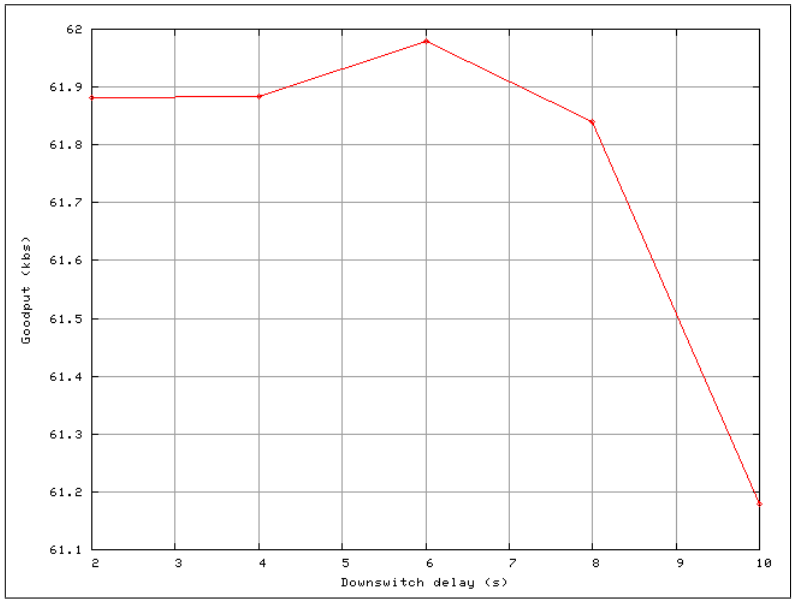

Figure 7.10: Impact of the inactivity timer on the goodput

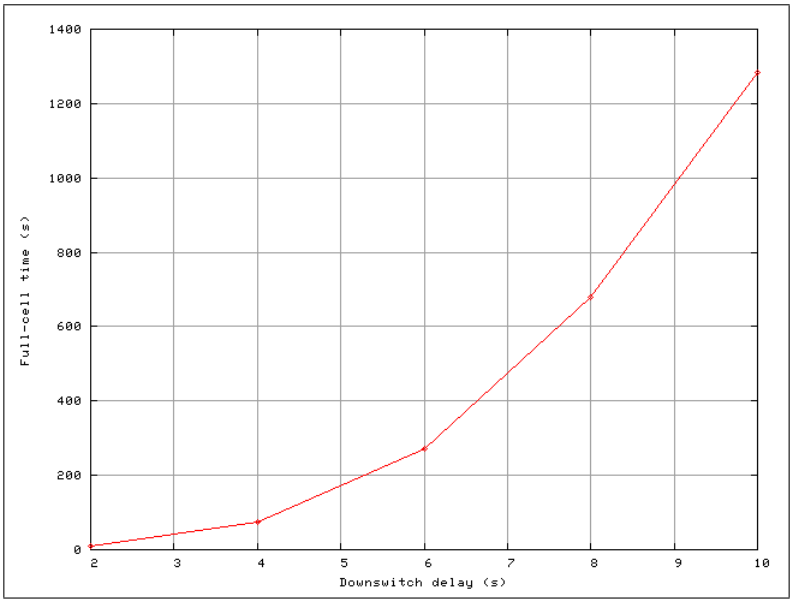

Figure 7.11: Impact of the inactivity timer on the full-cell time

7.4 Conclusion

In this chapter, we saw the impact of different RLC parameters with different versions of TCP, using the FACH/DCH switch in the case of WEB traffic. The goodput was taken as the best performance indicator. In general, SACK gave the worst performance of the tested protocols. There is no advantage of using a selective repeat mechanism in case of WEB traffic with local retransmissions. Indeed, there are only a few packets to transmit, so the CWND stays small, and when a loss occurs there are only a few unnecessary retransmissions when a Go-Back-N ARQ is used (like in Tahoe, Reno and Newreno)[30]. Moreover, a switch to a DCH can be caused by the Go-Back-N retransmissions if the upswitch threshold is higher than the packet size. The fact that SACK has to wait for a RTO to detect the loss of a retransmitted segment[22] also explain its poor performance.

The best results were achieved when maxDAT was high (6 in the simulations). However, increasing maxDAT to a higher value than 3 or 4 only gave really better results in the case of SACK. The upswitch threshold has a quite big impact on the goodput. The best threshold value depends on the used TCP protocol. SACK is very sensitive and its performance decreases very quickly as the threshold value grows. If SACK is used, the threshold must be as small as possible, to be sure that the first packet will trigger a switch to DCH. In the case of Tahoe, the performance stays quite stable if

Optimal values	
Parameter	Value
maxDAT	6
Upswitch threshold	500 bytes
Inactivity timer	8s

Table 7.4: Optimal values for the main parameters

the threshold is smaller than 1,000 bytes, and decreases rapidly afterwards. Reno and Newreno achieved the best performance when the threshold was equal to 1,000 bytes.

The performance following the inactivity timer is strongly dependent on the packet scheduler. In the implemented scheduler, an inactive connection cannot be pulled out of its DCH before the end of the inactivity timer. If its value is very large, the cell is often fully loaded, meaning that no DCH can be allocated to the connections that need it. These connections have to stay in FACH, which is a lot slower than DCH, and experience more congestion problems at the BS.

Therefore, the best parameters, whatever protocol is used, are a high maxDAT value (6 in these simulations), a threshold smaller than the average packet size (500 or 1,000 bytes in these simulations), and a not too long inactivity timer (smaller than 10 seconds in these simulations). Table 7.4 shows the values that gave optimal results for all the protocols.

For these parameter values, the TCP Tahoe, Reno and Newreno performance were similar. The goodput is 63.1 kbps, and a packet call takes about 2.7 or 2.8 seconds. The performance of SACK was a little bit worse, with a 62.8kbps goodput, and a packet call length of 3 seconds.

Chapter 8

Simulation results: Web traffic using the FACH/DCH switch, with hybrid inactivity timers

8.1 Introduction

In the last chapter, we saw that a large non-preemptive inactivity timer leads to a small goodput, as the cell can easily become fully loaded and UEs have therefore to transmit on the common channels. A way to avoid this problem may be to use a non-preemptive timer [8]. The chosen implementation uses a hybrid approach of the inactivity timer. When there is no more activity on the DCH, a short non-preemptive timer is set, during which no other user can use the reserved DCH. After this short time a preemptive timer is set, meaning that if the bandwidth is needed by another UE, the UE currently in DCH can be pulled out of it to give the bandwidth to the user who needs it (see Chapter 5 for more details). Please note that if a DCH has to be released before an allocation could take place, the allocation last 1s instead of 0.5s since there must be a de-allocation followed by an allocation.

First, the simulated scenario and the considered parameters will be given. Then, the results from the simulation runs will be explained.

8.2 Simulated scenarios, parameters and performance metrics

Most of the parameters are the same as the ones used in Chapter 7. However, the simulation topology and parameters will be shortly reminded here. The scenario considers 30 users downloading WEB data in one cell. The users begin using the common channels, and when the RLC buffer grows above the upswitch threshold, an allocation to a DCH is triggered. Each simulation is ran 3 times for 4,000 seconds, with each time a different seed for the random variables used in the traffic and error generators. Then, the results were averaged to get statistically sound values.

The results analyzed in Chapter 7 (Figure 7.7) showed that a large maxDAT value implicates a high goodput. Therefore, maxDAT was set to 10 in all the simulations, and the impact of maxDAT has not been shown here. The parameters of interest are the upswitch buffer threshold (between 500 Bytes and 1,500 Bytes) and the two different inactivity timers. Figure 8.1 summarizes the hybrid preemptive/non-preemptive scheme described in Section 5.2.

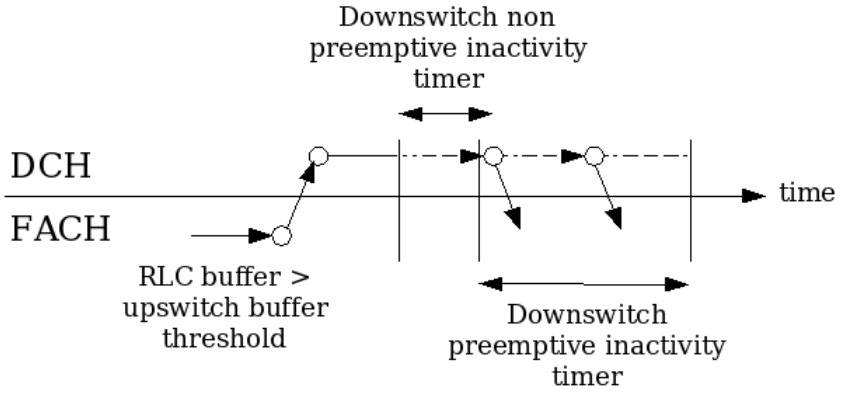

Figure 8.1: DCH allocation with hybrid inactivity timers.

The first goal of the non-preemptive inactivity timer is to help a UE keeping its DCH if the connection is not closed. It is used to avoid that each time a RTO occurs the DCH is released. The preemptive inactivity timer is used in order to keep the DCH between several consecutive connections if no other UE needs it. Therefore, the non-preemptive timer values (between 0 and 8 seconds) are smaller than the preemptive ones (between 4 and 16 seconds).

Main parameters	
Parameter	Value
Number of users	30
Traffic	WEB
Switching time	0.5s
Error rate	multistate
Pr(Error)	0.055
maxDAT	10
Upswitch buffer threshold	500, 750, 1,000, 1,250, 1,500 Bytes
Intermediate inactivity timer	0, 2, 4, 6, 8s
Final inactivity timer	4, 8, 12, 16s
TCP version	Tahoe, Reno, Newreno, Sack
Simulation length	4,000s
Number of simulations	3
FACH bandwidth	32kbs
FACH TTI	10ms
RACH bandwidth	16kbs
RACH TTI	20ms
DCH downlink bandwidth	64kbs
DCH downlink TTI	10ms
DCH uplink bandwidth	32kbs
DCH downlink TTI	10ms
RLC PDU size	40bytes
RLC POLL timeout	170ms
RLC buffer size	100kB
Max cell downlink bandwidth	800kbps

Table 8.1: Summary of the main parameters

The TCP versions that are described in this chapter are Tahoe, Reno, Newreno and SACK. Table 8.1 summarizes the values of the parameters for the simulations. The parameters used in the WEB traffic generation are listed in Section 5.3. The switching delay between the FACH and the DCH was taken to be 0.5 seconds, in order to stay close to the values listed in [8] and [12]. The other parameters are the same as described in Chapter 7.

8.3 Results

Please note that due to unexpected technical problems, it has sometimes been impossible to validate the discussions of the following sections. Therefore, some explanations here are only hypotheses to explain the protocol's behavior. First, the impact of the upswitch threshold on the goodput will be shown. A description of the TCP retransmissions evolution follows. These first parts will only describe averaged results since all the TCP versions behave the same. Finally, the effect of the newly implemented inactivity timers on the goodput will be described for different upswitch threshold values and different TCP versions. These last sections will not show the TCP Newreno results since its behavior was almost the same as Reno's.

8.3.1 Upswitch threshold and goodput

Figure 8.2 shows the impact of the upswitch threshold on the goodput. These results have to be compared with Section 7.3.4 and with some results of Appendix A. In these past results, the maximum goodput difference following the threshold is 1.8% in Section 7.3.4 and 1.5% in Appendix A. This difference is not really big, and becomes even smaller (1%) in the last set of simulations. As seen before, the largest threshold achieves the worst performance, but the optimal threshold is not necessarily the smallest, depending on the TCP version. The general behavior is the same as seen previously: Reno and Newreno achieves the best performance for a 1,000 Bytes threshold and Sack seems to give the best goodput for a small threshold value (see Section 7.3.4).

The Tahoe behavior and performance are almost the same than Reno and Newreno since most of the connections keep using a DCH. Sack achieved a good performance with a large threshold, compared to the simulations with only a non-preemptive inactivity timer. A reason could be that it is easy to keep a DCH for several consecutive connections if no other UE needs it, thanks to the quite large preemptive inactivity timer. Therefore, the allocations caused by a fast retransmission mechanism like Reno, Newreno and Tahoe provide a small advantage when a large preemptive inactivity timer is used. Moreover, the good performance of Sack on the common channels

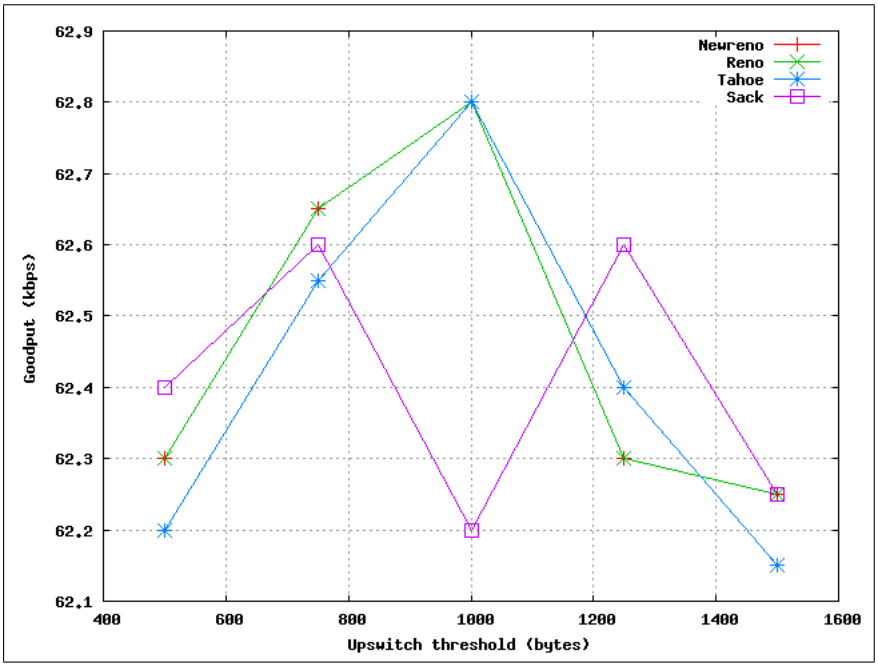

Figure 8.2: Goodput vs upswitch threshold

(Chapter 6) is a great advantage against the other versions since a lot of data is sent on FACH because of the large upswitch threshold.

8.3.2 TCP retransmissions

Figure 8.3 shows the average number of retransmissions per connection, for different inactivity timers values. The first interesting finding to notice is that the number of retransmissions and of RTO is a lot smaller than for the similar simulations of Chapter 7 (0.077 instead of 1.76 in Table 7.3). The proportion of RTOs per retransmission is similar, but the fact that most of the allocations last 0.5s instead of 1s avoids most of the RTOs, and then most of the retransmissions.

The main conclusion we can draw from this figure is that the number of retransmissions increase exponentionally with the non-preemptive inactivity timer. Indeed, as seen in Chapter 7, a high value of non-preemptive inactivity timer keeps the cell full and prevents a lot of connections to send data on a DCH. A lot of data has thus to be sent through the common channels, causing congestion and retransmissions (see Chapter 6). Figure 8.4 helps to confirm that there are less DCH allocations if the non-preemptive timer is high. However, the same trend can be observed about the preemptive timer,

because it allows a UE to keep its DCH between several consecutive connections if the bandwidth is not needed. Less allocations (and then less RTOs) can thus allow the same number of connections to use a DCH. Figure 8.4 also shows that the benefits of sparing DCH allocations (allocation delays) are bigger when the non-preemptive inactivity time is large.

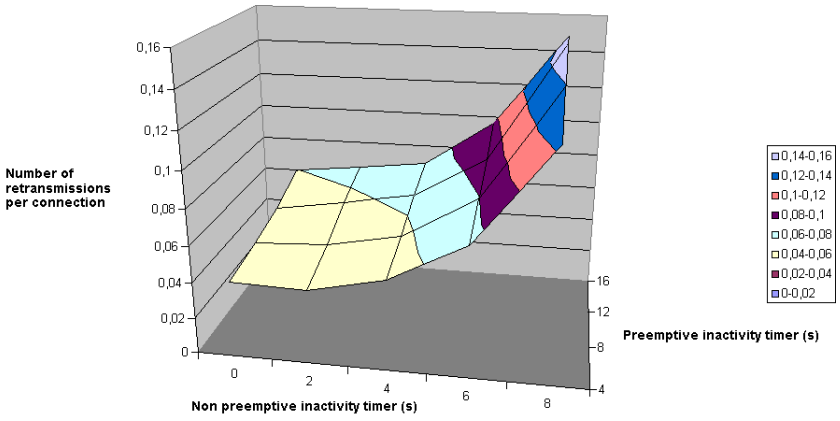

Figure 8.3: Retransmissions per connection, following the inactivity timers (average of all TCP versions).

Figure 8.5 shows the number of times a RTO occurred per retransmitted packet. This can be a good indication of the cause of the the retransmissions, three duplicate ACKs or to a RTO. Following this figure, the proportion of retransmissions due to a RTO is decreasing very fast when the non-preemptive inactivity timer increases. The reason is the same as seen above: a high non-preemptive timer causes a lot of UE to send data on FACH. The congestion problem on the common channel then provokes a lot of packet drops. These drops are mainly detected through three duplicate ACKs since the packets following the dropped one could arrive without problems to the destination UE.

Finally, another trend can be seen on Figureq 8.3 and 8.5. The number of retransmissions increases a little bit and the proportion of RTO decreases with the preemptive inactivity timer. There are less RTOs with a big preemptive timer because several connections can use the same DCH, and there are less DCH allocations. Moreover, if the switching time grows (it is probably 1s instead of 0.5s if the preemptive timer is large), the number of TCP packets that can arrive in the RLC buffer during the switching time increases too, causing more packets to be retransmitted because of the same RTO. Indeed, when a RTO occurs, all the packets sent after the one that caused the RTO are sent again in the case of Reno, Newreno and Tahoe. In the case

of Sack, the behavior is the same. If a RTO occurs because of the allocation time, no TCP packet following the one that caused the RTO could arrive to the destination and generate a Selective ACK. Therefore, all the following TCP packets are retransmitted too.

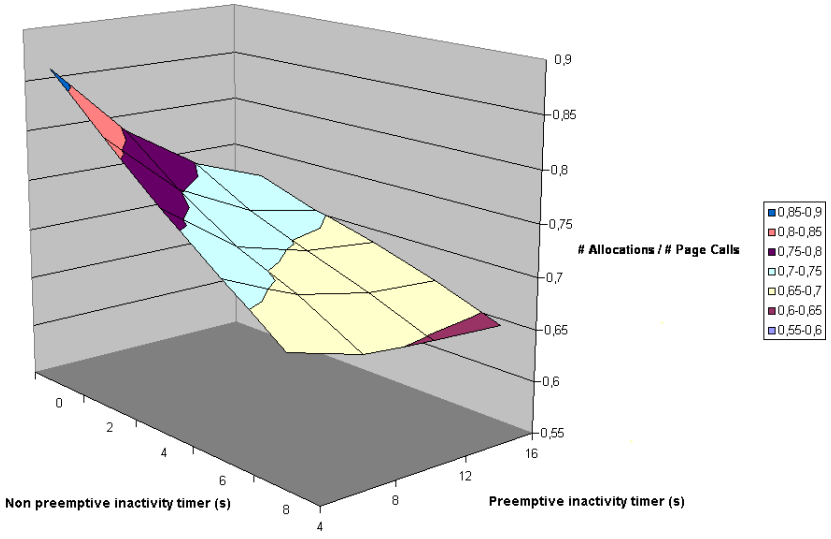

Figure 8.4: Number of DCH allocation per connection, following the inactivity timers (average of all TCP versions).

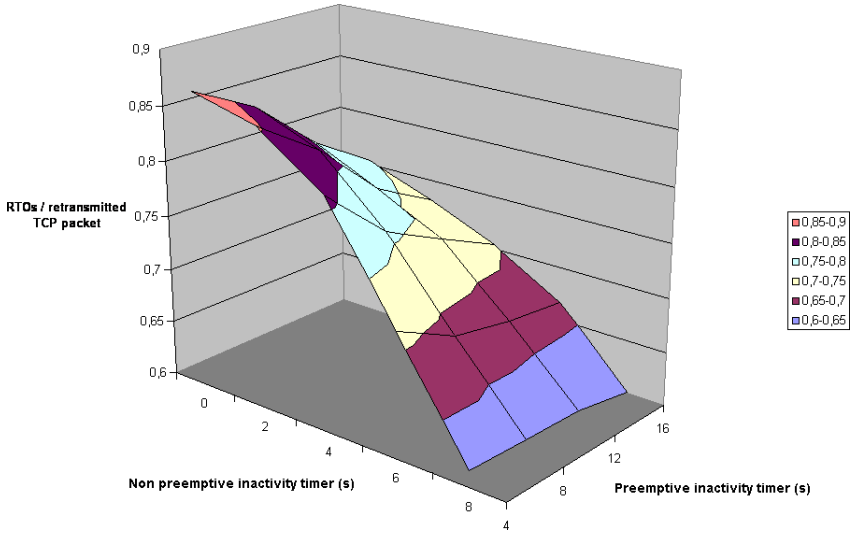

Figure 8.5: RTOs per retransmission, following the inactivity timers (average of all TCP versions).

8.3.3 Goodput (upswitch threshold is 750 Bytes)

When the upswitch threshold is set to 750 Bytes, a DCH is easily allocated. In this case, the behavior of TCP Tahoe, Reno and Newreno is almost the same. Figures 8.6 and 8.7 show the impact of the non-preemptive inactivity timer and of the preemptive inactivity timer on the goodput, for Reno and Tahoe. The behavior of the two described TCP versions is almost exactly the same. Two different trends can be seen on these graphs.

First, the goodput tends to decrease if the non-preemptive inactivity timer is large. The reason is the same than in Chapter 7 (Figure 7.10). A high non-preemptive inactivity timer causes the cell to be often fully loaded, meaning that some connections were unable to obtain the DCH they needed. Therefore, these connections had to transmit in FACH that is not "congestion-free" (see Chapter 6) and a lot slower than a DCH.

Second, a hole can be seen in the graphs with a small non-preemptive timer and a large preemptive timer. A small non-preemptive timer causes a connection not to keep its DCH for more than a few seconds, even when the inactivity is due to a RTO. On the other hand, as the upswitch threshold is small, a DCH can easily be allocated and there might be a lot of short unnecessary and/or repeated DCH allocations. Moreover, a large preemptive inactivity timer causes a lot of DCH allocations to last 1 second instead of

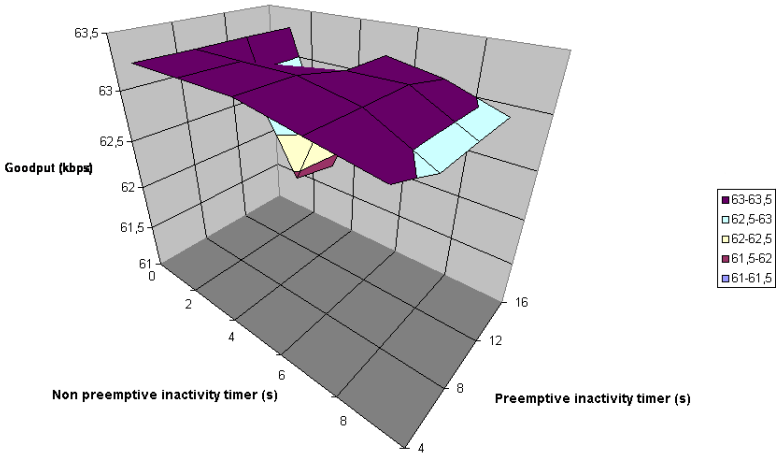

Figure 8.6: Goodput and hybrid inactivity timers, Reno, 750Bytes thresh.

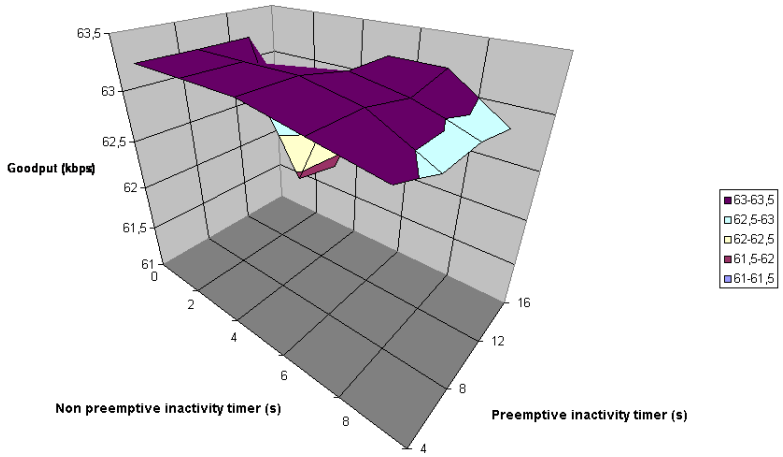

Figure 8.7: Goodput and hybrid inactivity timers, Tahoe, 750Bytes thresh.

0.5 second if a UE has to be downswitched before the new allocation.

8.3.4 Goodput (upswitch threshold is 1,000 Bytes)

When the upswitch threshold is set to 1000 Bytes, a DCH can be allocated but the very small connections can send data on the common channels without having to wait for a DCH. If the time needed to switch is too large compared to the amount of transferred data, it is better not to trigger a switch to a DCH (see Section 8.3.1). Figures 8.8 and 8.9 show the impact of the non-preemptive inactivity timer and of the preemptive inactivity timer on the goodput, respectively for Reno and Tahoe.

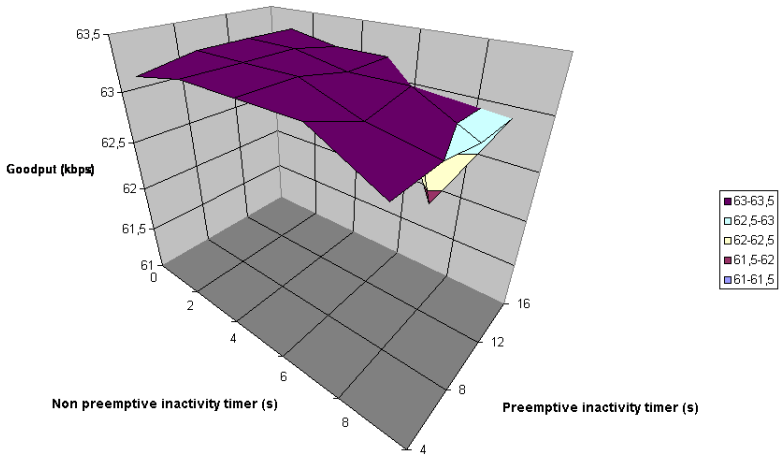

Figure 8.8: Goodput and hybrid inactivity timers, Reno, 1,000Bytes thresh.

The goodput decrease seen with a 750Byte upswitch threshold, a small non-preemptive and a large preemptive inactivity timer disapeard. The fact that the upswitch threshold is larger makes the DCH allocation more difficult and avoids the several unnecessary and/or repeated DCH allocations that happened when the upswitch threshold was 750 Bytes.

The goodput decrease following the non-preemptive inactivity timer is still a trend when the threshold equals to 1,000 Bytes. However, this small trend becomes a real dip in the graph for Reno and Tahoe, when the preemptive inactivity timer is large. As seen before, a large preemptive timer causes a lot of allocations to last 1 seconds instead of 0.5 second. Moreover,

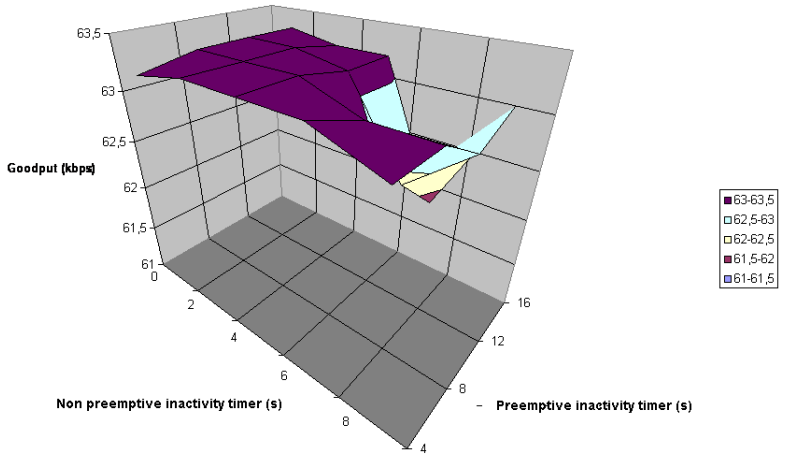

Figure 8.9: Goodput and hybrid inactivity timers, Tahoe, 1,000Bytes thresh.

a threshold of 1,000 Bytes means that if a user asks for a DCH, a lot of data is waiting and the DCH is really needed. Then, the fact that the allocation will last 1 second instead of 0.5 second may be a serious handicap. In the case of Reno, the goodput decrease occurs only for the highest preemptive timer value (16 seconds), while it occurs even for smaller values of preemptive timer in the case of Tahoe. Reno is less sensitive to the preemptive timer than Tahoe because of its fast recovery mechanism that helps the UE to fill the RLC buffer and therefore to easily trigger an upswitch to a DCH (see Sections 7.3.4 and 8.3.1).

8.3.5 Goodput (upswitch threshold is 1,500 Bytes)

With a very large upswitch threshold as 1500 Bytes, it becomes more difficult to trigger a switch to a DCH. In general, a too large upswitch threshold achieves a bad performance for the Tahoe and Reno TCP versions, as seen in Figures 7.8 and 7.9. There is no more real trend to explain the protocol behavior. However, this section will attempt to explain the obtained results. Figures 8.10 and 8.11 show the impact of the non-preemptive inactivity timer and of the preemptive inactivity timer on the goodput, respectively for Reno and Tahoe.

If the two inactivity timers are small, all the connections that needs a DCH will probably get it with a 0.5s switching time since there are only a

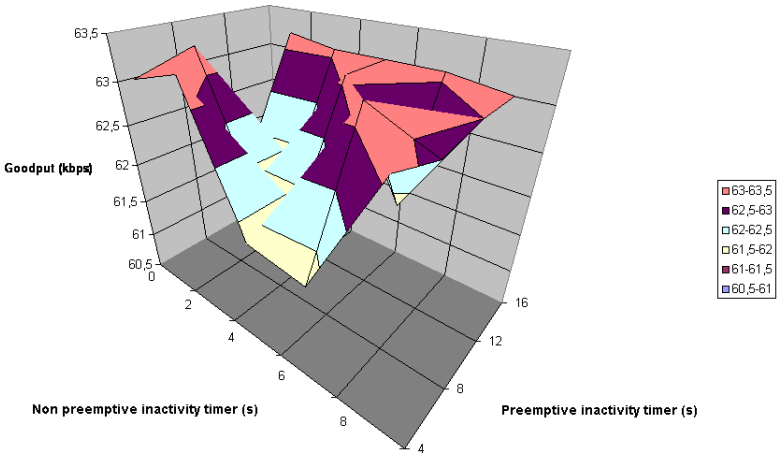

Figure 8.10: Goodput and hybrid inactivity timers, Reno, 1,500 Bytes thresh.

few DCH allocations and the UE never keeps their DCH between the connections. When the two timers grows, it begins to be more difficult to obtain a DCH, and the switch often last 1s instead of 0.5s. Moreover, as the sum of the two timers stays quite small compared to the reading time between the connections (less than 12s against 30s, see Section 5.3), the UE rarely keeps a DCH for several consecutive connections. The goodput is then quickly reduced. When the sum of the two inactivity timers increases, the goodput grows. This might be because if a DCH allocation is difficult, lasts a long time and/or is impossible because the cell is fully loaded, it is better to keep its DCH as long as possible when allocated. As for the smaller upswitch threshold values, Reno and Tahoe have almost the same behavior.

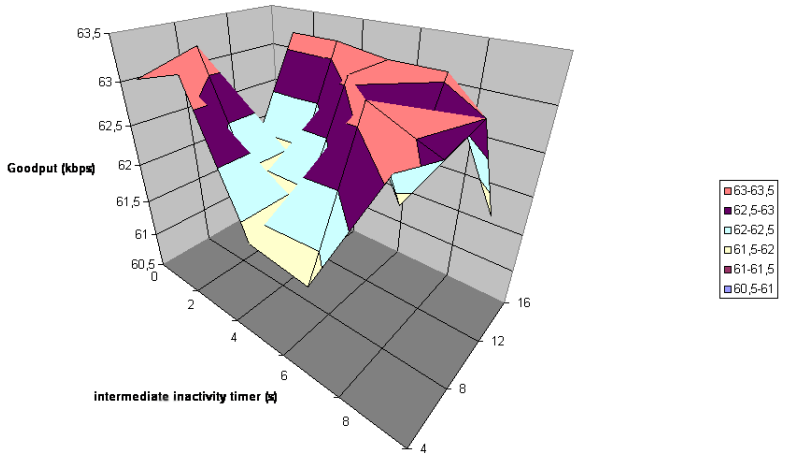

Figure 8.11: Goodput and hybrid inactivity timers, Tahoe, 1,500 Bytes thresh.

8.3.6 Goodput while using Sack

As the behavior of the Sack protocol was often quite different from the other versions, it seemed more appropriate to gather the informations about Sack in a separate section. This section will show the impact of the two inactivity timers on the Sack goodput, for different upswitch threshold values.

Figure 8.12 shows the impact of the inactivity timers on the goodput, when a 750 Bytes upswitch threshold is used with Sack. The two trends here are the same than for Reno and Tahoe (see Section 8.3.3). However, the goodput decrease following the non-preemptive inactivity timer is really small since Sack performs well when sending data on the common channels.

Figure 8.13 shows that the behavior of Sack is more unpredictable when the upswitch threshold is 1,000 Bytes. It seems to be a really bad parameter value for Sack. With this threshold value, the TCP connections requesting a DCH tend to fill the RLC buffer and to trigger a switch to a DCH. The problem is that if there is congestion on the common channels, Sack will reduce its sending rate and retransmit only the missing TCP packet. Therefore a connection that would use a DCH with the other TCP versions will trigger a switch to a DCH with Sack too, but later than the other protocols, as these ones can fill the RLC buffer with their retransmitted packets. More data is then transmitted on the common channels with Sack. The goodput

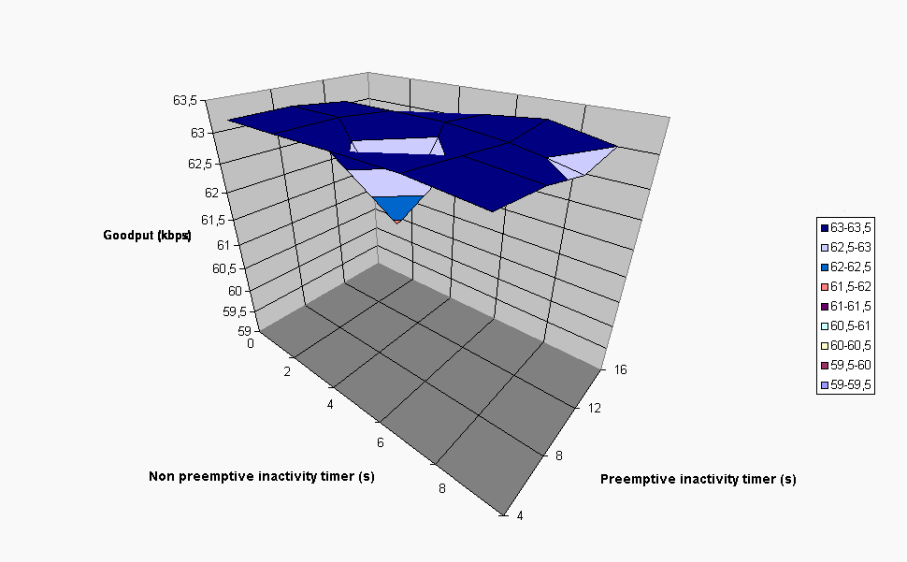

Figure 8.12: Goodput and hybrid inactivity timers, Sack, 750 Bytes thresh.

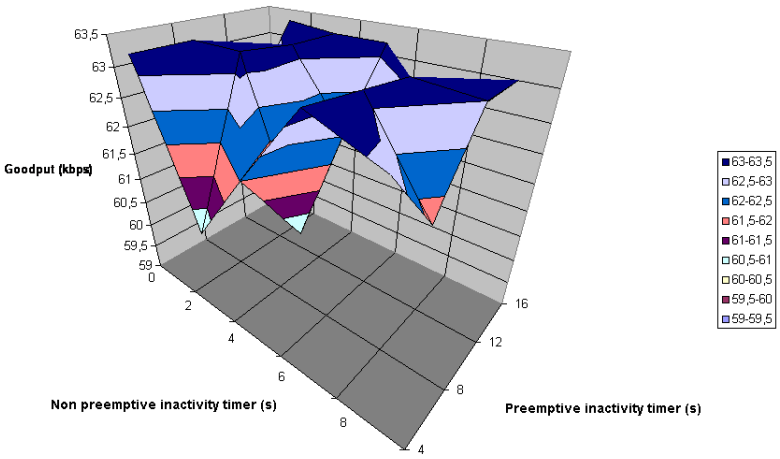

Figure 8.13: Goodput and hybrid inactivity timers, Sack, 1,000Bytes thresh.

performance thus strongly depends on the congestion of the common channels.

Figure 8.14 shows that with a large upswitch threshold (1,500 Bytes), Sack behavior is very foreseeable and follow a clear trend. With such a large threshold, it is really difficult for the UE using Sack to fill the RLC buffer in order to obtain a DCH. The main idea is that as it is difficult to trigger a switch, a UE has to try to keep it as long as possible. However, if another UE needs it, it should be possible to de-allocate it. The preemptive inactivity should be large in order to fulfill this objective. Moreover, as seen on Figures 8.4 and 8.15, the gain of DCH allocation thanks to a large preemptive timer is large if the non-preemptive timer is small, and reduces to null for bigger values. In this case, the DCH should therefore be kept longer in order to compensate the goodput losts because the cell was fully loaded and some data had to be sent on FACH.

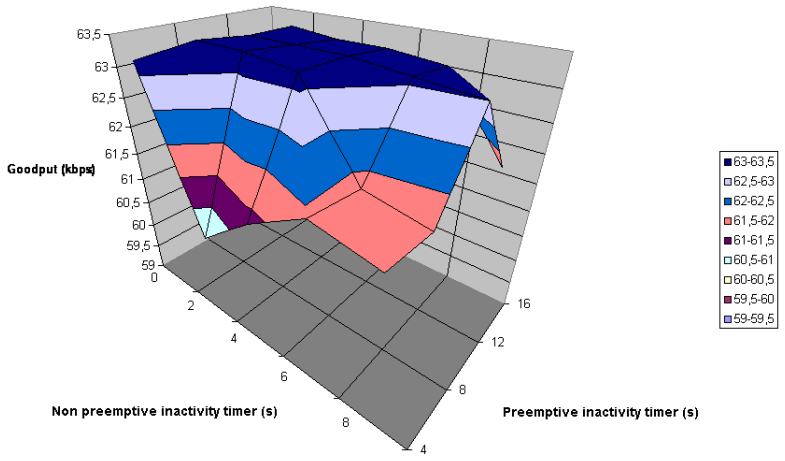

Figure 8.14: Goodput and hybrid inactivity timers, Sack, 1,500Bytes thresh.

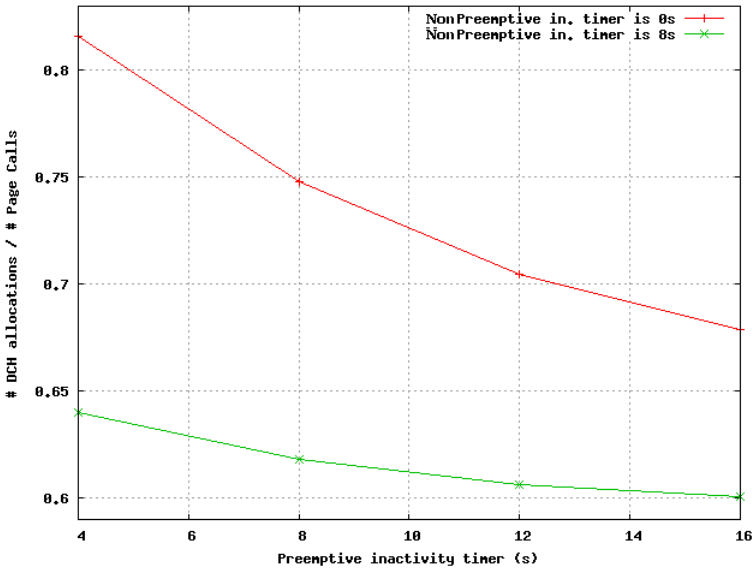

Figure 8.15: Number of DCH allocations per connection following the preemptive in. timer, 1,500 Bytes thresh.

8.4 Conclusion

This chapter described the impact of the upswitch RLC buffer threshold, and of a hybrid non-preemptive/preemptive inactivity timer mechanism. This analysis was done through simulations of web traffic download over one cell, using the possible switch between FACH and DCH. As in Chapter 7, the goodput was taken as the best performance indicator.

The conclusions of Chapter 7 and Appendix A are still valid in this last set of simulations. Apart from Sack that seems to prefer a small threshold, the best upswitch threshold value on the average is not the smallest but 1000 Bytes. However, it is necessary to take care of the inactivity timers values. For some of them, a threshold of 1,000 Bytes can deliver very poor performance. Therefore, a safe threshold value seems to be 750 Bytes, since most of the other parameters combinations achieve a maximal goodput and that all the TCP versions' behaviors seem to be foreseeable. The different TCP versions' behaviors are close to each other because with a small upswitch threshold, even Sack can easily trigger a switch to a DCH. In this case, the good values for the inactivity timers could be 0 or 2 seconds for the non-preemptive timer and 4 or 8 seconds for the preemptive timer. A too large non-preemptive timer causes the cell to be often fully loaded, and thus more data has to be sent on the common channels, while a big preemptive timer causes a lot of DCH allocation to last 1s instead of 0.5s.

For greater upswitch threshold values, the TCP Reno, Newreno and Tahoe behaviors are still foreseeable and almost the same. However, for a very large threshold (1,500 Bytes in this chapter), the goodput is a lot more sensitive to the inactivity timer values.

Sack always behave differently than the other TCP versions. The main reason is that it is more difficult for Sack to trigger a switch to a DCH by filling the RLC buffer with retransmissions than for the other protocols, especially for bigger threshold values. Moreover, Sack is a little bit more efficient than the other protocols when using the common channels, thanks to its Selective Acknowledgement and Selective Repeat mechanism. The most unpredictable behavior of Sack could be seen with a 1,000 bytes upswitch threshold. With this kind of threshold, a connection that would use a DCH with the other TCP versions will trigger a switch to a DCH with Sack too, but later than the other protocols that can quickly fill the RLC buffer with their retransmitted packets. More data is then transmitted on the common channels, and the goodput performance thus strongly depends on the congestion of the common channels.

In general, Sack performed better with the hybrid inactivity timer approach than with the single non-preemptive inactivity timer, especially in case the upswitch threshold is large. The main reason is that the preemptive inactivity timer allows the UE to keep a DCH for a longer time and for several connections, sometimes avoiding Sack difficulties to fill the RLC buffer.

Chapter 9

Conclusions

9.1 Conclusion

This report investigated the downlink performance of different TCP protocols over UMTS common and dedicated channels, using a RLC in Acknowledged Mode. The study was performed with the help of simulations using the discrete event simulator NS2 [16] and its UMTS extension EURANE [17]. Some new functionalities had to be implemented to make this report possible.

The first set of simulations considered two FTP sessions over FACH. When there was only one user, it was shown that the number of RLC retransmissions (maxDAT) had a great impact on the TCP goodput. The goodput grew with maxDAT until a certain threshold after which it stay constant. When maxDAT was smaller than this threshold, TCP packets were dropped because of the BLER. These losses were understood as a congestion notification by the TCP sender. In this case, Sack and Newreno achieved the highest goodput. When maxDAT was larger than this threshold, Vegas gave the best results because its congestion control took the RTT as a congestion indicator, while the other versions tended to have their outstanding window only limited by the advertised window. A large outstanding window lead to a large RTT and therefore to a small goodput.

The congestion problem at the BS was only shown in the case of a very high maxDAT. For limited maxDAT values, the bandwidth was not fairly shared between the connections because of the DropTail policy at the BS. When a second user began to transmit on FACH, the total goodput decreased a little bit with all the TCP versions. The Vegas goodput was still higher than the others when congestion occurred, and its RLC throughput was significantly smaller. Indeed, TCP Vegas was aware of the congestion through its enhanced congestion control and reduced its CWND accordingly.

The second set of simulations considered 30 users downloading WEB traffic, with a possible switch between the common and dedicated channels. The downswitch mechanism used a single non-preemptive inactivity timer and the cell had limited total bandwidth. The goodput was taken as the best quality indicator. Simulations showed that a big maxDAT achieves the best performance. The different TCP versions obtained different goodputs for small maxDAT values, but when maxDAT grew the results were similar.

The last set of simulations was the same as the previous one with an hybrid non-preemptive/preemptive downswitch mechanism. When no activity is detected on a DCH, a short non-preemptive timer is set, during which no other user can use the reserved DCH. After this short time, a preemptive timer is launched, meaning that if the bandwidth is needed by another UE, the UE currently in DCH can be pulled out of it to give the bandwidth to the user who needs it. After these two inactivity timers, the DCH is released anyway.

Tahoe, Reno and Newreno performed almost the same for all the upswitch threshold values. Moreover, their behavior seemed to be stable and predictable, especially if the threshold was not too large. Sack performed differently, and the less foreseeable behavior could be seen with a 1,000 bytes threshold. With this kind of threshold, a connection that would use a DCH with the other TCP versions would trigger a switch to a DCH with Sack too, but later than the other protocols. More data were then transmitted on the common channels, and the goodput performance thus usually decreased compared to the other TCP versions and strongly depended on the congestion of the common channels.

In general, Sack performed better with the hybrid inactivity timer approach than with the single non-preemptive inactivity timer, especially when the upswitch threshold was large. The main reason is that the preemptive inactivity timer allows the UE to keep a DCH for a longer time and for several consecutive connections, sometimes avoiding Sack difficulties to trigger a switch to a DCH.

For both inactivity timer approaches, the RLC upswitch buffer threshold had a big impact on the goodput. The best threshold value depended on the used TCP protocol and on the other parameters. However, with a threshold higher than the average packet size, the goodput decreased significantly. Sack was the most sensitive TCP version and needed an upswitch threshold as small as possible, while TCP Reno and Newreno gave in general a better performance with a higher threshold. Indeed, if the packet size was smaller than the threshold, a high number of retransmissions could trigger

an upswitch to a DCH. The number of retransmissions was always smaller in the case of Sack, since it used a Selective Repeat mechanism.

In general, it was surprising to note that TCP Sack did not achieve a better performance than the other TCP versions. These poor results come from the used WEB traffic, that keeps the outstanding window small [30], and from the FACH/DCH switch, that is triggered faster if the packet size is smaller than the upswitch threshold. The fact that SACK had to wait for a RTO to detect the loss of a retransmitted segment could explain the bad performance too.

9.2 Future work

Due to technical problems, all the possible simulations could not be launched. However, the scripts and source code are ready to be launched if the project is resumed. This last section is intended to give some ideas of what could be done if someone decides to continue the project.

The problems encountered with TCP FACK and TCP Vegas should be corrected, particularly because FACK is intended to better recover from episodes of heavy loss than Sack [23]. Moreover, Vegas achieved the best performance on the data transfer over FACH (see Chapter 6).

A new packet scheduler may be implemented in order to be able to dynamically allocate different bandwidths depending on the free cell bandwidth [12].

Simulations with a different traffic type can be done in order to see if the behavior of the TCP versions (especially Sack) changes with larger CWND. A new FTP traffic generator following the 3GPP specifications [9] is already implemented and just needs to be launched. However, the simulations using this kind of traffic must last longer than for WEB traffic since there are less connections and the file sizes are larger.

It can be interesting to run simulations with an increased TCP's Initial Window. It is a well known improvement for wireless communications [12], but the effect will not necessarily be positive if a switch is triggered to a DCH. If the first packets sent are retransmitted because of the allocation time, it will lead to unnecessary retransmissions, and the CWND will be decreased anyway before new packets could be sent.

Finally, it could be interesting to make some data transfer simulations using the HS-DSCH. In order to do this, the RLC maxDAT implementation

has to be adapted for this type of channel.

Bibliography

[1] Claffy K., Greg Miller, Kevin Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone", 1998

[2] Jon Postel, "Transmission Control Protocol, DARPA Internet Program Protocol Specification", RFC 793, 1981

[3] T. Chahed, A-F. Canton, S-E. Elayoubi, "End-to-end TCP Performance in W-CDMA/UMTS", IEEE, 2003

[4] Robert Bestak, Philippe Godlewski, Philippe Martins, "RLC buffer occupancy when using a TCP connection over UMTS", ENST, IEEE, 2002

[5] Ni Zhang, "Simulation-based investigation of TCP file transfers over UMTS", 2003

[6] Fabienne Lefevre, Guillaume Vivier, "Optimizing UMTS Link Layer Parameters for a TCP Connection", IEEE, 2001

[7] Anthony Lo, Geert Heijenk, Cezar Bruma, "Performance of TCP over UMTS Common and Dedicated Channels", IST, 2003

[8] B. Prabhu, E. Altman, K. Avrachenkov, J. Dominguez, "A Simulation Study of TCP Performance over UMTS Downlink", INRIA, 2004.

[9] 3GPP, TR 25.876 V1.6.1, "Multiple-Input Multiple Output in UTRA", 2004

[10] 3GPP, TR 21.905 v4.4.0, "Vocabulary for 3GPP Specifications (release 4)", 2001

[11] Korhonen J., "Introduction to 3G Mobile Communication (2nd edition)", 2003.

[12] Holma H., Toskala A. ,(...), "WCDMA For UMTS - Radio Access For Third Generation Mobile Communications (3rd edition)", Wiley, 2004.

[13] Stevens W. R., "TCP/IP Illustrated, Volume 1: The protocols", Addison-Wesley Professional Computing Series, 1994

[14] James Kurose, Keith Ross, "Analyse structuree des reseaux (2e edition)", Pearson Education, 2003

[15] Laurent Schumacher, "Teleinformatique et reseaux m.a. (INFO2231)", Lecture, University of Namur (FUNDP), 2004.

[16] Kevin Fall, Kannan Varadhan, "The ns Manual", VINT Project, 2003

[17] "EURANE User Guide (Release 1.3)", SEACORN Project, 2004

[18] Jae Chung, Mark Claypool, "NS by Example", Worcester Polytechnic Institute

[19] Oumer Teyeb, Jeroen Wigard, "FACE: Future Adaptive Communication Environment, Deliverable 2.1: Emulation of TCP Performance Over WCDMA", Department of Communication Technology, AAU, Aalborg, 2003

[20] Olivier Hynderick, Sven Raes, "TCP performance over WCDMA", Institut d'Informatique, FUNDP, Namur, 2004

[21] Lawrence Brakmo, Sean O Malley, Larry Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", ACM SIGCOMM, 1994

[22] Kevin Fall, Sally Floyd, "Simulation-based Comparison of Tahoe, Reno and SACK TCP", Lawrence Berkeley National Laboratory, 1996

[23] Matt Mathis, Jamshid Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM, 1996

[24] Charitakis Yannis, Papadakis Charalampos, "End-to-End Congestion Avoidance in TCP: an Introduction", Department of Teleinformatics, KTH

[25] Almudena Konrad, Ben Y. Zhao, Anthony D. Joseph, Reiner Ludwig, "A Markov-Based Channel Model Algorithm for Wireless Networks"

[26] Oumer Teyeb, Malek Boussif, "Emulation Based Performance Investigation of FTP File Downloads over UMTS Dedicated Channels", 2004

[27] Juan Rendon, Ramon, Ferrus, Ferran Casadevall, Anna Sfairopoulo, "Experimental analisys of TCP behaviour over a downlink UMTS channel under different scheduling strategies", June 2004

[28] Antonis Alexiou, Christos Bouras, Vaggelis Igglesis, "Performance Evaluation of TCP over UMTS Transport Channels", 2004

[29] 3GPP, TS 25.322 v5.4.0, "Radio Link Control (RLC) protocol specification, release 5", 2003

[30] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, R. H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements", IEEE, 1998

[31] Llorenc Cerda and Olga Casals, "Study of the TCP Unfairness in a Wireless Environment", IEEE, 2001

[32] Theodore Faber, "ACC: Using Active Networking to Enhance Feedback Congestion Control Mechanisms", IEEE, 1998

[33] "Tcptrace Official Homepage", http://www.tcptrace.com

[34] "Tcpdump Official Homepage", http://www.tcpdump.org

Appendix A

Performance comparison of TCP versions when switching between UMTS common and dedicated channels

Submitted for presentation at Globecom 2005.

Appendix B

Optimum number of RLC Retransmissions for Best TCP Performance in UTRAN

Submitted for presentation at PIMRC 2005.